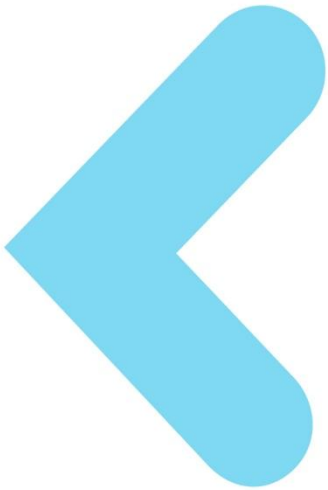
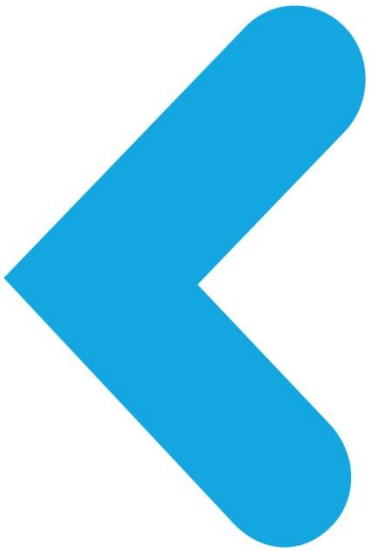
TRUSTONIC



‹t-base API Documentation

API Level 3

API Doc.



PREFACE

VERSION HISTORY

Version	Date	Modification
1.0	February 1st, 2013	First version for API Level 1
2.0	March 18th, 2013	Updated for API Level 2
3.0	November 18th, 2013	Updated for API Level 3

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This document specifies the API for developing Trusted Applications and Client Applications:

<t-base provides two distinct set of APIs for developers:

- The <t-base Legacy API to develop legacy Trusted Applications using the tlAPI and the mcClient API. Note that the mcClient API is also available at the kernel level.
- The GlobalPlatform API to develop Trusted Applications as defined by GlobalPlatform using the GlobalPlatform TEE Client and Internal APIs.

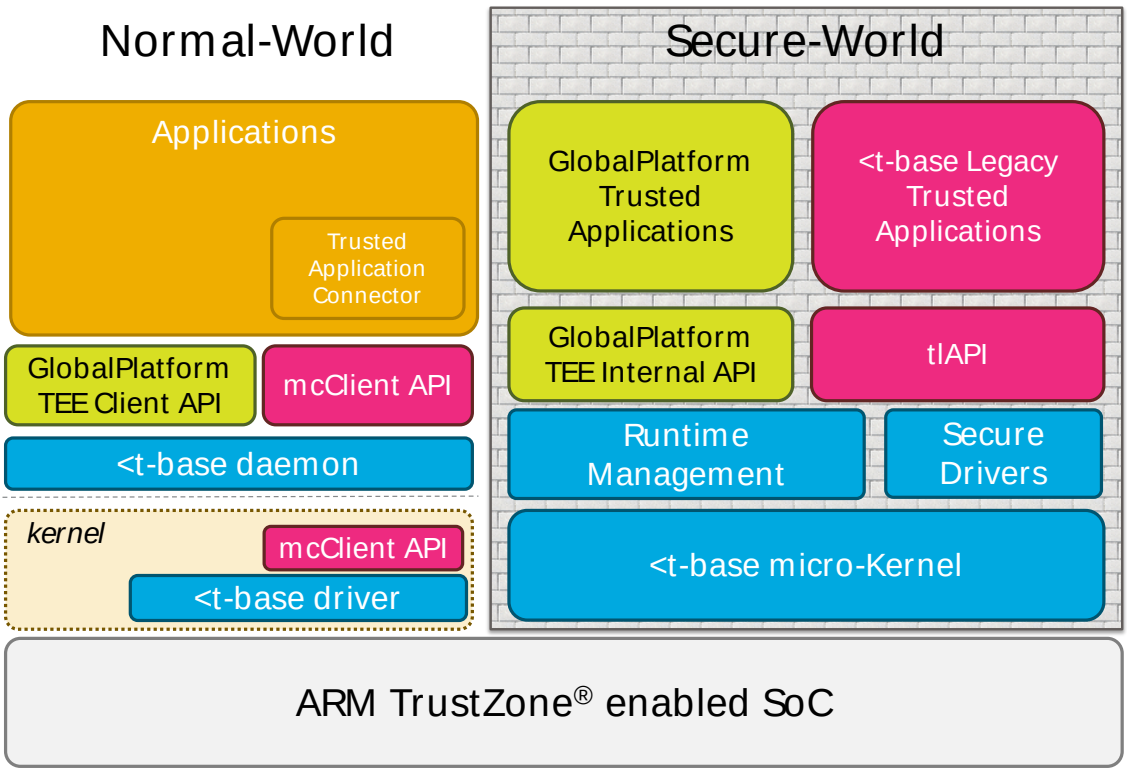


Figure 1: <t-base API Overview.

For introduction and guidance on how to develop for <t-base, please refer to the `<t-base Developer Guide`.

2 API VERSION HISTORY

API Level	Change
Level 1	First <t-base API
Level 2	Added mcOpenTrustlet() function Added tlApiGetVirtMemType() function Added tlApiIsNwdBufferValid() function mcOpenSession() is deprecated mcMallocWsm() is deprecated mcFreeWsm() is deprecated Added tlApiDeriveKey() function
Level 3	Added Trusted User Interface API Added DRM API Added GlobalPlatform API

3 tlApi

The tlApi is the <t-base legacy API for developing Trusted Applications

3.1 HEADER FILE

The header file for the tlApi is "`TlApi.h`".

```
#include "TlApi.h"
```

3.2 GENERIC CONSTANTS

3.2.1 Error Codes

Constant Name	Value	Definition
E_TLAPI_BUFFER_INCORRECT_TYPE	0x00000110	Passed buffer is not of correct type.
E_TLAPI_BUFFER_TOO_SMALL	0x00000107	Buffer is too small.
E_TLAPI_COM_ERROR	0x0000010f	Internal communication error.
E_TLAPI_COM_WAIT	0x0000010e	Waiting for a notification failed.
E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE	0x00000302	Algorithm is not available for the caller.
E_TLAPI_CR_ALGORITHM_NOT_SUPPORTED	0x00000303	The intended algorithm usage is not supported.
E_TLAPI_CR_HANDLE_INVALID	0x00000301	No running session is associated with this handle value or caller has not permission to access the session associated with this handle value.
E_TLAPI_CR_INCONSISTENT_DATA	0x00000306	Inconsistency of data was determined.
E_TLAPI_CR_OUT_OF_RESOURCES	0x00000307	No more additional resources available.
E_TLAPI_CR_WRONG_INPUT_SIZE	0x00000304	Input data (message or data to be encrypted or decrypted) is too short or too long.
E_TLAPI_CR_WRONG_OUPUT_SIZE	0x00000305	Provided Output buffer is of wrong size.
E_TLAPI_DRV_INVALID_PARAMETERS	0x00000203	Driver parameters invalid.
E_TLAPI_DRV_NO_SUCH_DRIVER	0x00000202	Unknown driver, bad driver ID.
E_TLAPI_DRV_UNKNOWN	0x00000201	Unspecified driver error.
E_TLAPI_INVALID_INPUT	0x00000105	Input data is invalid.

E_TLAPI_INVALID_PARAMETER	0x0000010b	Invalid parameter.
E_TLAPI_INVALID_RANGE	0x00000106	If address/pointer has invalid range
E_TLAPI_INVALID_TIMEOUT	0x00000108	The chosen timeout value was invalid.
E_TLAPI_NOT_IMPLEMENTED	0x00000101	Function not yet implemented.
E_TLAPI_NULL_POINTER	0x0000010a	Null pointer.
E_TLAPI_SO_CONTEXT_KEY_FAILED	0x00000404	Derivation of context key failed.
E_TLAPI_SO_CONTEXT_NOT_PERMITTED	0x00000409	Unwrap does not permit this context.
E_TLAPI_SO_DELEGATED_NOT_PERMITTED	0x00000408	Unwrap does not permit delegated objects.
E_TLAPI_SO_WRONG_CHECKSUM	0x00000403	Wrong secure object checksum
E_TLAPI_SO_WRONG_CONTEXT	0x00000401	Illegal (unsupported) secure object context.
E_TLAPI_SO_WRONG_LIFETIME	0x00000405	Illegal (unsupported) secure object lifetime.
E_TLAPI_SO_WRONG_PADDING	0x00000402	Wrong padding of secure object.
E_TLAPI_SO_WRONG_TYPE	0x00000407	Illegal (unsupported) secure object type.
E_TLAPI_SO_WRONG_VERSION	0x00000406	Illegal (unsupported) secure object version.
E_TLAPI_TIMEOUT	0x00000109	Timeout expired.
E_TLAPI_UNALIGNED_POINTER	0x0000010d	Passed pointer is not word-aligned.
E_TLAPI_UNKNOWN	0x00000102	Unknown error during TlApi usage.
E_TLAPI_UNKNOWN_FUNCTION	0x00000104	Function not known.
E_TLAPI_UNMAPPED_BUFFER	0x0000010c	Specified buffer is not entirely mapped in Trusted Application address space.
TLAPI_OK	0x00000000	TlApi return values. Returns on successful execution of a function.

Table 1: Trusted Application API Error Codes

3.2.2 Macros

Macro Name	Definition
TLAPI_ERROR_DETAIL(ecode)	Get detail part of error code.
TLAPI_ERROR_MAJOR(ecode)	Get MAJOR part of error code.
TLAPI_ERROR_MAJOR_CODE(ecode)	Get MAJOR_CODE part of error code.

TLAPI_ERROR_MAJOR_COMPONENT(ecode)	Get MAJOR_COMPONENT part of error code.
TLAPI_INFINITE_TIMEOUT ((uint32_t)(-1))	Wait infinite time for a response of the MC.
TLAPI_NO_TIMEOUT	Do not wait for a response of the MC.
MC_SO_SIZE_F22(plainLen, encryptedLen)	Calculates the total size of a secure object. plainLen is the length of plain text part within secure object. encryptedLen is the length of encrypted part within secure object (excl. hash, padding). Returns the total (gross) size of the secure object or 0 if given parameters are illegal or would lead to a secure object of invalid size.

Table 2: Trusted Application API Macros

3.2.3 Other Constants

Constant Name	Value	Definition
MC_SO21_HASH_SIZE	24	Size of hash used for secure object v2.1.
MC_SO21_RND_SIZE	9	Size of random used for secure objects v2.1.
MC_SO22_HASH_SIZE	32	Size of hash used for secure object v2.2.
MC_SO22_RND_SIZE	16	Size of random used for secure objects v2.2.
MC_SO_ENCRYPT_BLOCK_SIZE	16	Block size of encryption algorithm used for secure objects.
MC_SO_HASH_SIZE	32	Size of hash used for secure objects v2.
MC_SO_MAX_PADDING_SIZE	(MC_SO_ENCRYPT_ BLOCK_SIZE)	Maximum number of ISO padding bytes.
MC_SO_PAYLOAD_MAX_SIZE	1000000	Maximum size of the payload (plain length + encrypted length) of a secure object.

Table 3: Other Constants

3.3 GENERIC TYPES

3.3.1 Enumerations and types

3.3.1.1 mcSoContext_t

Secure object context.

A context defines which key to use to encrypt/decrypt a secure object.

Enumeration:

- MC_SO_CONTEXT_TLT Trusted Application context.
- MC_SO_CONTEXT_SP Service provider context.

- MC_SO_CONTEXT_DEVICE Device context.
- MC_SO_CONTEXT_DUMMY Dummy to ensure that enum is 32 bit wide.

3.3.1.2 mcSoLifeTime_t

Secure object lifetime.

A lifetime defines how long a secure object is valid.

Enumeration:

- MC_SO_LIFETIME_PERMANENT SO does not expire.
- MC_SO_LIFETIME_POWERCYCLE SO expires on reboot (coldboot).
- MC_SO_LIFETIME_SESSION SO expires when Trusted Application is closed.
- MC_SO_LIFETIME_DUMMY Dummy to ensure that enum is 32 bit wide.

3.3.1.3 mcSoType_t

Secure object type.

Enumeration:

- MC_SO_TYPE_REGULAR Regular secure object.
- MC_SO_TYPE_DUMMY Dummy to ensure that enum is 32 bit wide.

3.3.1.4 mcSpid_t

```
typedef uint32_t mcSpid_t ;
```

Service provider Identifier type.

Constant Name	Value	Definition
MC_SPID_FREE	0xFFFFFFFF	SPID value used as free marker in root containers.
MC_SPID_RESERVED	0	Reserved UUID.
MC_SPID_SYSTEM	0xFFFFFFFE	UUID for system applications.

Table 4 mcSpid_t Constants

3.3.2 mcUuid_t - Universally Unique Identifier.

Universally Unique Identifier (UUID) according to ISO/IEC 11578.

```
typedef struct {
    uint8_t value[16]; /**< Value of the UUID. */
} mcUuid_t, *mcUuid_ptr;
```

This structure can be initialized with 3 different values:

Constant Name	Value	Definition
MC_UUID_FREE	{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }	UUID value used as free marker in service provider containers.
MC_UUID_RESERVED	{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }	Reserved UUID.
MC_UUID_SYSTEM	{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE }	UUID for system applications.
MC_SUID_LEN	16	Length of SUID.

Table 5: mcUuid_t Constants

3.3.3 mcSoHeader_t

```
typdef struct
    uint32_t type;
    uint32_t version;
    mcSoContext_t context;
    mcSoLifeTime_t lifetime;
    tlApiSpTrustletId_t producer;
    uint32_t plainLen;
    uint32_t encryptedLen;
} mcSoHeader_t;
```

The fields of the structure are:

- type: Type of secure object.
- version: Secure object version.
- context: Secure object context.
- lifetime: Secure object lifetime.
- producer: Producer Trusted Application id.
- plainLen: Length of unencrypted user data (after the header).
- encryptedLen: Length of encrypted user data (after unencrypted data, excl. checksum and excl. padding bytes).

A secure object header introduces a secure object. Layout of a secure object:

```
+--------+------------------+------------------+--------+--------+
| Header |   plain-data     |  encrypted-data  |  hash  | random |
+--------+------------------+------------------+--------+--------+
/--------/---- plainLen ----/-- encryptedLen --/-- 32 --/-- 16 --/
/----------------- toBeHashedLen --------------/
                            / toBeEncryptedLen /
/--------------------------- totalSoSize -----------------------/
```

Secure object header in v2.1 is:

```
+--------+------------------+------------------+--------+--------+---------+
| Header |   plain-data     |  encrypted-data  |  hash  | random | padding |
+--------+------------------+------------------+--------+--------+---------+
/--------/---- plainLen ----/-- encryptedLen --/-- 24 --/--- 9 --/- 0..15 -/
/----------------- toBeHashedLen --------------/
                            / toBeEncryptedLen /
/--------------------------- totalSoSize ---------------------------------/
```

Secure object header in v2.0 is:

```
+--------+------------------+------------------+--------+---------+
| Header |   plain-data     |  encrypted-data  |  hash  | padding |
+--------+------------------+------------------+--------+---------+
/--------/---- plainLen ----/-- encryptedLen --/-- 32 --/- 1..16 -/
/----------------- toBeHashedLen --------------/
                             /---------- toBeEncryptedLen --------/
/--------------------------- totalSoSize -----------------------/
```

3.3.4 mcSuid_t

```
typedef struct {
    uint32_t    sipId;
    suidData_t  suidData;
} mcSuid_t;
```

Universally Unique Identifier (UUID) according to ISO/IEC 11578.

The fields of the structure are:

- sipId : Silicon Provider ID to be set during build
- uint8_t suidData [12]: Value of the UUID.

3.3.5 suidData_t

```
/** Length of SUID. */
#define MC_SUID_LEN    16

typedef struct {
    uint8_t data[MC_SUID_LEN - sizeof(uint32_t)];
} suidData_t;
```

Platform specific device identifier (serial number of the chip).

3.3.6 tlApiRsaKey_t

```
typedef struct {
    tlApiLongInt_t  exponent;
    tlApiLongInt_t  modulus;
    tlApiLongInt_t  privateExponent;
    struct {
        tlApiLongInt_t Q;
        tlApiLongInt_t P;
        tlApiLongInt_t DQ;
        tlApiLongInt_t DP;
        tlApiLongInt_t Qinv;
    } privateCrtKey;

} tlApiRsaKey_t;
```

The fields of the structure are:

- exponent: Pointer to public exponent.
- modulus: Modulus (if public key present).
- privateExponent: Private exponent (if private key present).
- Q: Pointer to prime q (if private CRT key present).
- P: Pointer to prime p (if private CRT key present).
- DQ: Pointer to DQ := d mod (Q-1) (if private CRT key present).
- DP: Pointer to DP := d mod (P-1) (if private CRT key present).
- Qinv: Pointer to Q inverse Qinv := Q^{-1} mod P (if private crt key present).

3.3.7 tlApiKey_t

```
typedef union {
    tlApiSymKey_t *symKey;
    tlApiRsaKey_t *rsaKey;
} tlApiKey_t;
```

Union of key structure pointers. Enables generic interfaces.

The fields of the structure are:

- symKey: Pointer to symmetric key.
- rsaKey: Pointer to RSA key.

3.3.8 tlApiSymKey_t

```
typedef struct {
    uint8_t *key;
    uint32_t len;
 } tlApiSymKey_t;
```

Symmetric key structure.

The fields of the structure are:

- key: Pointer to the key.
- len: Byte length of the key.

3.3.9 tlApiKeyPair_t

```
typedef union {
    tlApiRsaKey_t *rsaKeyPair;*
} tlApiKeyPair_t;
```

Symmetric key structure.

The fields of the structure are:

- rsaKeyPair: Pointer to RSA key structure.

3.3.10 tlApiLongInt_t

```
typedef struct {
    uint8_t *value;
    uint32_t len;
} tlApiLongInt_t;
```

Symmetric key structure.

The fields of the structure are:

- value: Pointer to value. Byte array in big endian format.
- len: Byte length of value.

3.3.11 tlApiPlatformInfo_t

```
typedef struct {
    uint32_t size;
    uint32_t manufacturerdId;
    uint32_t platformVersion;
    uint32_t platformInfoDataLength;
    uint8_t  platformInfoData[];
} tlApiPlatformInfo_t;
```

The platform information structure returns manufacturer specific information about the hardware platform.

The fields of the structure are:

- size: size of the structure.
- manufacturerdId: Manufacturer ID provided by Trustonic.
- platformVersion :Version of platform
- platformInfoDataLength : Length of manufacturer specific platform information
- platformInfoData: Manufacturer specific platform information data.

3.3.12 tlApiSpTrustletId_t

```
typedef struct {
    mcSpid_t spid;
    mcUuid_t uuid; } tlApiSpTrustletId_t;
```

Service provider Trusted Application id.

The fields of the structure are:

- spid: Service Provider ID.
- uuid: Trusted Application UUID.

3.3.13 tlApiSymKey_t

```
typedef struct {
    uint8_t *key;
    uint32_t len;
} tlApiSymKey_t;
```

Symmetric key structure.

The fields of the structure are:

- key: Pointer to the key.
- len: Byte length of the key.

3.4 INTER-WORLD COMMUNICATION

<t-base provides Trusted Applications with a set of functions for inter-world communication.

Communication is based on shared memory buffers and notifications without a message payload.

Message-formatting is application specific. Messages are interchanged on world shared memory buffers that the Trustlet Connector (TLC) specifies in `mcOpenSession()` and `mcMap()`. The <t-base driver and <t-base OS set up this shared memory buffer between TLC and Trusted Application and forward notifications between the two.

3.4.1 tlApiNotify()

```
_TLAPI_EXTERN_C tlApiResult_t tlApiNotify (void);
```

Notify the Trusted Application Connector about changes in the Trusted Application Communication Interface (TCI) message buffer.

Trusted Applications must use the **tlApiNotify()** function to inform the <t-base TEE that the session channel contains information to be processed by its Trusted Application Connector. Trusted Applications must not make assumptions at what point in time the information will be processed by the Trusted Application Connector. It is up to <t-base TEE to decide when to inform the Trusted Application Connector. **tlApiNotify()** returns when the notification is processed and the Trusted Application can continue.

Returns:

- TLAPI_OK if notification has been issued successfully.

3.4.2 tlApiWaitNotification

```
_TLAPI_EXTERN_C tlApiResult_t tlApiWaitNotification (uint32_t timeout)
```

Wait for a notification of the NWd.

Trusted Applications use the **tlApiWaitNotification()** to wait for a notification by their Trusted Application Connector. Calling **tlApiWaitNotification()** will block the Trusted Application until a notification dedicated to the Trusted Application session arrives. Depending on the underlying hardware platform, a wait timeout may be provided. If the waiting timeout is not supported by the platform E_TLAPI_INVALID_TIMEOUT is returned.

Parameters:

- timeout: time in milliseconds to wait (TLAPI_NO_TIMEOUT => direct return, TLAPI_INFINITE_TIMEOUT => wait infinitely

Returns:

- TLAPI_OK if notification has been received.

3.5 CRYPTOGRAPHY

<t-base provides Trusted Applications with access to cryptographic primitives.

Depending on the platform, cryptographic functions are implemented in a software library or using a hardware cryptographic engine. Depending on the used cryptographic driver, certain functions may or may not exist.

3.5.1 Constants

Constant Name	Value	Definition
AF_CIPHER	(1U << 24)	Algorithm ID is composed of group flags and a consecutive number. The upper 16bits are used for grouping, whereas the lower 16bits are available to distinguish algorithms within each group. Algorithm type flags.
AF_CIPHER_AES	(1U << 16)	Subgroups of cipher algorithms.
AF_SIG_DES	(1U << 16)	Subgroups of signature algorithms.
CR_SID_INVALID	0xffffffff	Invalid crypto session id returned in case of an error.

Table 6 Cryptography Constants

3.5.2 Types

3.5.2.1 tlApiCrSession_t

```
typedef uint32_t tlApiCrSession_t;
```

Handle of a crypto session.

3.5.3 Enumerations

3.5.3.1 tlApiCipherAlg_t

List of Cipher algorithms:

- AES ciphers
- Triple-DES ciphers
- DES ciphers
- RSA ciphers

An algorithm in this list is to be interpreted as a combination of cryptographic algorithm, paddings, block sizes and other information.

Enumerator Name	Definition
TLAPI_ALG_AES_128_CBC_NOPAD	AES with key size 128 in CBC mode, no padding.
TLAPI_ALG_AES_128_CBC_ISO9797_M1	AES with key size 128 in CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_128_CBC_ISO9797_M2	AES with key size 128 in CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

TLAPI_ALG_AES_128_CBC_PKCS5	AES with key size 128 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_CBC_PKCS7	AES with key size 128 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_ECB_NOPAD	AES with key size 128 in ECB mode, no padding.
TLAPI_ALG_AES_128_ECB_ISO9797_M1	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_128_ECB_ISO9797_M2	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_128_ECB_PKCS5	AES with key size 128 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_ECB_PKCS7	AES with key size 128 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_CTR_NOPAD	AES with key size 128 in CTR mode, no padding.
TLAPI_ALG_AES_256_CBC_NOPAD	AES with key size 256 in CBC mode, no padding.
TLAPI_ALG_AES_256_CBC_ISO9797_M1	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_CBC_ISO9797_M2	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_CBC_PKCS5	AES with key size 256 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_256_CBC_PKCS7	AES with key size 256 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_ECB_NOPAD	AES with key size 256 in ECB mode, no padding.
TLAPI_ALG_AES_256_ECB_ISO9797_M1	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_ECB_ISO9797_M2	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_ECB_PKCS5	AES with key size 256 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_256_ECB_PKCS7	AES with key size 256 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_CTR_NOPAD	AES with key size 256 in CTR mode, no padding.
TLAPI_ALG_AES_128_CBC_PKCS5	AES with key size 128 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_CBC_PKCS7	AES with key size 128 in CBC mode, padding according to the PKCS#7 scheme.

TLAPI_ALG_AES_128_ECB_NOPAD	AES with key size 128 in ECB mode, no padding.
TLAPI_ALG_AES_128_ECB_ISO9797_M1	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_128_ECB_ISO9797_M2	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_128_ECB_PKCS5	AES with key size 128 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_ECB_PKCS7	AES with key size 128 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_CTR_NOPAD	AES with key size 128 in CTR mode, no padding.
TLAPI_ALG_AES_256_CBC_NOPAD	AES with key size 256 in CBC mode, no padding.
TLAPI_ALG_AES_256_CBC_ISO9797_M1	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_CBC_ISO9797_M2	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_CBC_PKCS5	AES with key size 256 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_256_CBC_PKCS7	AES with key size 256 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_ECB_NOPAD	AES with key size 256 in ECB mode, no padding.
TLAPI_ALG_AES_256_ECB_ISO9797_M1	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_ECB_ISO9797_M2	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_ECB_PKCS5	AES with key size 256 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_256_ECB_PKCS7	AES with key size 256 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_CTR_NOPAD	AES with key size 256 in CTR mode, no padding.
TLAPI_ALG_AES_128_CBC_PKCS5	AES with key size 128 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_CBC_PKCS7	AES with key size 128 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_ECB_NOPAD	AES with key size 128 in ECB mode, no padding.
TLAPI_ALG_3DES_CBC_ISO9797_M1	Triple DES in outer CBC mode, padding according to the ISO 9797 method 1 scheme.

TLAPI_ALG_3DES_CBC_ISO9797_M2	Triple DES in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_3DES_CBC_NOPAD	Triple DES in outer CBC mode, no padding.
TLAPI_ALG_3DES_CBC_PKCS5	Triple DES in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_DES_CBC_ISO9797_M1	DES in CBC mode or triple DES in outer CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_DES_CBC_ISO9797_M2	DES in CBC mode or triple DES in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_DES_CBC_NOPAD	DES in CBC mode or triple DES in outer CBC mode, no padding.
TLAPI_ALG_DES_CBC_PKCS5	DES in CBC mode or triple DES in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_DES_ECB_ISO9797_M1	DES in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_DES_ECB_ISO9797_M2	DES in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_DES_ECB_NOPAD	DES in ECB mode, no padding.
TLAPI_ALG_DES_ECB_PKCS5	DES in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_RSA_ISO14888	RSA, padding according to the ISO 14888 scheme.
TLAPI_ALG_RSA_NOPAD	RSA, no padding.
TLAPI_ALG_RSA_PKCS1	RSA, padding according to the PKCS#1 (v1.5) scheme.

Table 7 tlApiCipherAlg Enumeration

3.5.3.2 tlApiCipherMode_t

Enumerator Name	Value	Definition
TLAPI_MODE_ENCRYPT	0	Encryption mode.
TLAPI_MODE_DECRYPT	1	Decryption mode.

Table 8 tlApiCipherMode_t Enumeration

3.5.3.3 tlApiKeyPairType_t

List of Key Pair types.

Enumeration:

TLAPI_RSA RSA public and RSA normal / CRT private key.

3.5.3.4 tlApiMdAlg_t

List of Message Digest algorithms.

Enumerator Name	Value	Definition
TLAPI_ALG_MD2	AF_MD | 1	Message Digest algorithm MD2.
TLAPI_ALG_MD5	AF_MD | 2	Message Digest algorithm MD5.
TLAPI_ALG_SHA1	AF_MD | 3	Message Digest algorithm SHA1.
TLAPI_ALG_SHA256	AF_MD | 4	Message Digest algorithm SHA256.

Table 9 tlApiMdAlg_t Enumeration

3.5.3.5 tlApiRngAlg_t

List of Random Data Generation algorithms.

Enumerator Name	Definition
TLAPI_ALG_SECURE_RANDOM	Random data which is considered to be cryptographically secure.
TLAPI_ALG_PSEUDO_RANDOM	Pseudo random data, most likely a returning pattern.

Table 10 tlApiRngAlg_t Enumeration

3.5.3.6 tlApiSigAlg_t

List of Signature algorithms.

An algorithm in this list is to be interpreted as a combination of cryptographic algorithm, paddings, block sizes and other information.

Enumerator Name	Definition
TLAPI_ALG_DES_MAC4_NOPAD	4-byte CBC-MAC (most significant 4 bytes of the last encrypted block) using DES in CBC mode or triple DES in outer CBC mode, with no padding.
TLAPI_ALG_DES_MAC4_PKCS5	4-byte CBC-MAC (most significant 4 bytes of the last encrypted block) using DES in CBC mode or triple DES in outer CBC mode, with PKCS#5 padding.
TLAPI_ALG_DES_MAC8_ISO9797_1_M2_ALG3	8-byte MAC using a 2-key DES3 key according to ISO9797-1 MAC algorithm 3 with method 2 (also EMV'96, EMV'2000), where input data is padded using method 2 and the data is processed as described in MAC Algorithm 3 of the ISO 9797-1 specification.
TLAPI_ALG_DES_MAC8_ISO9797_M1	8-byte CBC-MAC using DES in CBC mode or triple DES in outer CBC mode, following ISO9797-1 MAC with method 1.
TLAPI_ALG_DES_MAC8_ISO9797_M2	8-byte CBC-MAC using DES in CBC mode or triple DES in outer CBC mode, following ISO9797-1 MAC with method 1.

TLAPI_ALG_DES_MAC8_NOPAD	8-byte CBC-MAC using DES in CBC mode or triple DES in outer CBC mode, with no padding.
TLAPI_ALG_DES_MAC8_PKCS5	8-byte CBC-MAC using DES in CBC mode or triple DES in outer CBC mode, with PKCS#5 padding.
TLAPI_ALG_HMAC_SHA_256	HMAC following the steps found in RFC 2104 using SHA-256 (SHA-2) as the hashing algorithm.
TLAPI_ALG_HMAC_SHA1	HMAC following the steps found in RFC 2104 using SHA-1 as the hashing algorithm.
TLAPI_SIG_RSA_SHA_ISO9796	20-byte SHA-1 digest, padded according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, encrypted using RSA.
TLAPI_SIG_RSA_SHA_ISO9796_MR	20-byte SHA-1 digest, padded according to the ISO9796-2 specification and encrypted using RSA.
TLAPI_SIG_RSA_SHA_PKCS1	20-byte SHA-1 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA256_PSS	RSASSA-PSS according to PKCS#1 v2, Content digest SHA-256, MGF SHA-256.
TLAPI_SIG_RSA_SHA1_PSS	RSASSA-PSS according to PKCS#1 v2, Content digest SHA-1, MGF SHA-1.

Table 11 tlApiSigAlg_t Enumeration

3.5.3.7 tlApiSigMode_t

Main operation modes for signature.

Enumerator Name	Definition
TLAPI_MODE_SIGN	Signature generation mode.
TLAPI_MODE_VERIFY	Message and signature verification mode.

3.5.4 Functions

3.5.4.1 tlApiCipherDoFinal

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherDoFinal (
      tlApiCrSession_t sessionHandle,
      const uint8_t * srcData,
      size_t srcLen,
      uint8_t * destData,
      size_t * destLen)
```

Encrypts/decrypts the input data.

Processes data that has not been processed by previous calls to **tlApiCipherUpdate()** as well as data supplied in srcData. Completes the cipher session. Afterwards the session is closed and sessionHandle invalid. Input and output buffer may be the same (input data gets buffered block by block).

Parameters:

- sessionHandle: handle of a running Cipher session.
- srcData: reference to input data to be encrypted/decrypted.
- srcLen: byte length of input data to be encrypted/decrypted.
- destData: reference to result area.
- in,out destLen: [in] Byte length of buffer for output data. [out] Byte length of generated output.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session is invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]destLen is inconsistent with algorithm requirements.
- E_TLAPI_INVALID_INPUT if RSA modulus length is invalid.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if algorithm could not work with the input data (e.g. wrong padding).
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.2 tlApiCipherInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherInit (
      tlApiCrSession_t * pSessionHandle,
      tlApiCipherAlg_t alg,
      tlApiCipherMode_t mode,
      const tlApiKey_t * key)
```

Initializes a new cipher session.

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: output, reference to generated Cipher session handle (undefined in case of error).
- alg: see enum cipherMode_t.
- mode : TLAPI_MODE_ENCRYPT or TLAPI_MODE_DECRYPT.
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown or RSA cipher modulus length is zero.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.3 tlApiCipherInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherInitWithData (
      tlApiCrSession_t * pSessionHandle,
      tlApiCipherAlg_t alg,
      tlApiCipherMode_t mode,
      const tlApiKey_t * key,
      const uint8_t * buffer,
      size_t bufferLen)
```

Initializes a new cipher session.

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Note:

If the buffer/bufferLen contains invalid or inconsistent data (wrong length or length = 0, null, ...) a default value = 0 is taken and NO error is returned.

If the used algorithm doesn't use additional algorithm specific data, the given values are ignored and don't result in an error.

<t-base supports up to 2048 bit RSA keys.

Parameters:

- out pSessionHandle: reference to generated Cipher session handle (undefined in case of error).
- alg: See enum cipherMode_t.
- mode : TLAPI_MODE_ENCRYPT or TLAPI_MODE_DECRYPT.
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- buffer: reference to algorithm specific data (initial values).
- bufferLen: length of buffer containing algorithm specific data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown or RSA cipher modulus length is zero.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if key type doesn't match the algorithm.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.4 tlApiCipherUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * srcData,
```

```
    size_t srcLen,
    uint8_t * destData,
    size_t * destLen)
```

Encrypts/decrypts the input data.

Input data does not have to be multiple of block size. Subsequent calls to this method are possible. Unless one or several calls of this function have supplied sufficient input data no output is generated. Input and output buffer may be the same (input data gets buffered block by block).

Parameters:

- sessionHandle: handle of a running Cipher session.
- srcData: reference to input data to be encrypted/decrypted.
- srcLen: byte length of input data to be encrypted/decrypted.
- destData: reference to result area.
- in,out destLen: [in] Byte length of output buffer. [out] Byte length of generated output data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if during tlApiCipherInit() provided RSA padding is not available (function needs to check that input data does not exceed block size of RSA cipher).
- E_TLAPI_CR_WRONG_INPUT_SIZE if [in]srcLen is inconsistent with algorithm requirements.
- E_TLAPI_CR_WRONG_OUPUT_SIZE if [in]destLen is inconsistent with algorithm requirements.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if key type doesn't match the algorithm.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.5 tlApiCrAbort

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCrAbort (
    tlApiCrSession_t sessionHandle)
```

Aborts a crypto session.

Afterwards sessionHandle is not valid anymore.

Parameters:

- sessionHandle: Handle of session to be aborted.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided.

3.5.4.6 tlApiGenerateKeyPair

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGenerateKeyPair (
      tlApiKeyPair_t * keyPair,
      tlApiKeyPairType_t type,
      size_t len)
```

Generates a key pair.

The key components are generated according to requested type and length.

The caller has to set addresses in the key pair structure and initialize the public key exponent. Generated key components are written to those addresses. It is the responsibility of the caller to provide sufficient space and set length parameters of each of the buffer length elements in key structures appropriately. Length information of generated components will be overwritten with the actual length of the generated key pair structure elements.

For RSA the length value identifies the length of the modulus in bytes. The buffer for the generated modulus and private exponent (if present) must have room for at least len bytes.. If present, the buffers for the RSA CRT components P, Q, DP, DQ and QInv must have at least half the length of the modulus (rounded up). The generated modulus is a (len*8)-bit or (len*8-1)-bit number which is the product of two (len*4)-bit probable primes.

Note:

<t-base supports up to 2048 bit RSA keys.

Public exponent must have non-zero most significant byte. Also public exponent must be odd.

Parameters:

- in,out keyPair: Reference to key pair structure.
- type: See enum keyPairType_t.
- len: Requested byte length of keys.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers range.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_CR_WRONG_OUPUT_SIZE if provided buffer length of one of the buffers is too small.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.7 tlApiMessageDigestDoFinal

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestDoFinal (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      uint8_t * hash,
      size_t * hashLen)
```

Hashes the message.

Finishes the message digest session. Afterwards the session is closed and sessionHandle invalid.

Parameters:

- sessionHandle: Handle of a running session Message Digest session.
- message: Reference to message to be hashed.
- messageLen: Byte length of message.
- hash: Reference to generated hash.
- in,out hashLen: [in] Byte length of hash buffer. [out] Byte length of generated hash data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided. (session is not being closed).
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.8 tlApiMessageDigestInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestInit (
      tlApiCrSession_t * pSessionHandle,
      tlApiMdAlg_t algorithm)
```

Initializes a new Message Digest session.

A handle for the new session is generated. The algorithm type is set. If this method does not return with TLAPI_OK, then there is no valid handle returned. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: Reference to generated Message Digest session handle (undefined in case of error).
- Algorithm: See enum mdAlg_t.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.9 tlApiMessageDigestInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestInitWithData (
      tlApiCrSession_t * pSessionHandle,
      tlApiMdAlg_t alg,
      const uint8_t * buffer,
      const uint8_t lengthOfDataHashedPreviously[8])
```

Initializes a new Message Digest session.

A handle for the new session is generated. The algorithm type is set. Initializes a hash algorithm with a specified initialization vector. The initialization vector and the length of the previously hashed data needs to be provided to the function in big endian format. This may be used to calculate a part of the hash outside of the <t-base TEE and then finish the hash in the secure world. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: Reference to generated Message Digest session handle (undefined in case of error).
- alg: See mdAlg_t.
- buffer: Reference to previously calculated hash data.
- lengthOfDataHashedPreviously: Byte array in big endian format containing length of previously calculated hash.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.10 tlApiMessageDigestUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen)
```

Accumulates message data for hashing.

The message does not have to be blocksize aligned. Subsequent calls to this method are possible.

Parameters:

- sessionHandle: Handle of a running session Message Digest session.
- message: Reference to message to be hashed.
- messageLen: Byte length of input data to be hashed.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided. (session is not being closed).
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.11 tlApiRandomGenerateData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRandomGenerateData (
      tlApiRngAlg_t alg,
      uint8_t * randomBuffer,
      size_t * randomLen)
```

Generates random data.

Parameters:

- alg: See enum randomDataGenerationAlg_t.
- randomBuffer: Reference to generated random data.
- in,out randomLen: [in] Byte length of desired random length. [out] Byte length of generated random data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is unknown.
- E_TLAPI_DRV_UNKNOWN for any other errors.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.12 tlApiSignatureInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureInit (
      tlApiCrSession_t * pSessionHandle,
      const tlApiKey_t * key,
      tlApiSigMode_t mode,
      tlApiSigAlg_t alg)
```

Initializes a new signature session and returns the handle for the new session for further usage.

The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- out pSessionHandle: Reference to generated Signatures session handle (undefined in case of error).
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- Mode: TLAPI_MODE_SIGN or TLAPI_MODE_VERIFY.
- alg: see enum of algorithms.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed or key type doesn't match the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.13 tlApiSignatureInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureInitWithData (
      tlApiCrSession_t * pSessionHandle,
      const tlApiKey_t * key,
      tlApiSigMode_t mode,
      tlApiSigAlg_t alg,
      const uint8_t * buffer,
      size_t bufferLen)
```

Initializes a new signature session.

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Note:

If the buffer/bufferLen contains invalid or inconsistent data (wrong length or length = 0, null, ...) a default value = 0 is taken and NO error is returned.

If the used algorithm doesn't use additional algorithm specific data, the given values are ignored and don't result in an error.

For PSS signatures and verification bufferLen is interpreted as salt length, and buffer may be NULL.

For TLAPI_SIG_RSA_SHA_ISO9796_MR verify this function begins the verification sequence by recovering the message encoded within the signature itself and initializing the internal hash function. Therefore, the signature data needs to be provided in the buffer!

<t-base supports up to 2048 bit RSA keys.

Parameters:

- out pSessionHandle: Reference to generated Signatures session handle (undefined in case of error).
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- mode: TLAPI_MODE_SIGN or TLAPI_MODE_VERIFY.
- alg: see enum of algorithms.
- buffer: Reference to algorithm specific data like seed for hash or salt for PSS.
- bufferLen: Length of buffer containing algorithm specific data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed or key type doesn't match the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.

- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.14 tlApiSignatureSign

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureSign (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      uint8_t * signature,
      size_t * signatureLen)
```

Signs the message.

Finishes the signature session. Afterwards the session is closed and sessionHandle invalid. message pointer may be NULL if messageLen = 0.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be signed.
- messageLen: Byte length of message.
- in,out signature: Reference to generated signature.
- in,out signatureLen: [in] Byte length of signature buffer. [out] Byte length of generated signature.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided (session is not being closed).
- E_TLAPI_CR_OUT_OF_RESOURCES if required subsession could not be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]signatureLen is too short.
- E_TLAPI_INVALID_INPUT
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if the crypto library could not calculate a signature.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.15 tlApiSignatureUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen)
```

Accumulates data for a signature calculation.

Input data does not have to be multiple of blocksize. Subsequent calls of this method are possible. **tlApiSignatureSign()** or **tlApiSignatureVerify()** have to be called to complete the signature operation.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be signed/verified.
- messageLen: Byte length of message.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_WRONG_OUTPUT_SIZE
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if there was a problem with the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.16 tlApiSignatureVerify

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureVerify (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      const uint8_t * signature,
      size_t signatureLen,
      bool * validity)
```

Generates a signature for the supplied message and compares it to the supplied signature.

The generated signature is not presented to the caller. Finishes the signature session. Afterwards the session is closed and sessionHandle invalid. Message pointer may be NULL if messageLen = 0. Null pointer ex.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be verified.
- messageLen: Byte length of message.
- signature: Reference to signature to be verified.
- signatureLen: Byte length of signature.
- validity: Reference to verification result. TRUE if verified, otherwise FALSE.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided (session is not being closed).
- E_TLAPI_CR_OUT_OF_RESOURCES if required subsession could not be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]signatureLen is too short.
- E_TLAPI_INVALID_INPUT
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.5.4.17 tlApiDeriveKey

```
_TLAPI_EXTERN_C tlApiResult_t tlApiDeriveKey(
    const void *salt,
    size_t saltLen,
    void *dest,
    size_t destLen,
    mcSoContext_t context,
    mcSoLifeTime_t lifetime)
```

Derives a new key from the hardware master key. The key derivation function used by the implementation may vary across the implementations.

Different salt values provide different keys.

The resulting key is expanded to destLen bytes using RFC5869 expansion.

The derived key can be diversified between Trusted Applications or Service Providers depending on the context parameter.

The derived key can also be diversified between sessions or powercycles depending on the lifetime parameter.

Parameters:

- salt [in] Salt value for key derivation.
- saltLen [in] Length of salt value.
- dest [out] Resulting key.
- destLen [in] Length of desired key.
- context [in] Context for derived key. Possible values are:
  - MC_SO_CONTEXT_TLT: The key is diversified for each Trusted Application. This means a unique identifier of the calling Trusted Application is added to the salt.
  - MC_SO_CONTEXT_SP: The key is diversified for each Service Provider. This means a unique identifier of the Service Provider is added to the salt.
  - MC_SO_CONTEXT_DEVICE: The key is not diversified across Trusted Applications or Service Providers.
- lifetime [in] Lifetime for derived key. Possible values are:
  - MC_SO_LIFETIME_POWERCYCLE: the key is diversified for each powercycle. This means a unique identifier of the powercycle is added to the salt.
  - MC_SO_LIFETIME_SESSION: the key is diversified for each session of the Trusted Application. This means a unique identifier of the session is added to the salt.
  - MC_SO_LIFETIME_PERMANENT: the key is not diversified further across sessions or powercycles.

Returns:

TLAPI_OK if operation was successful.

3.6 SECURE OBJECTS

The secure object API provides integrity, confidentiality and authenticity for data that is sensitive but needs to be stored in untrusted (normal world) memory.

Secure objects provide device binding. Respective objects are only valid on a specific device.

There are two core operations of this API:

- wrap() which encloses user data in a secure object.
- unwrap() which extracts user data from a secure object.

The user data is divided into data that will remain as plain data and data that will be encrypted:

```
        +------------------+------------------------+
        |   plain-data     |  to-be-encrypted-data  |
        +------------------+------------------------+
        /-------------- user data ------------------/
        /- data that is    /-- data that will be
           not going to       encrypted. This data
           be encrypted       is encrypted by the
           and will remain    wrapper function ----/
           as cleartext ---/

        /---- plainLen ----/------ encryptLen -------/
```

A secure object looks like this:

```
+--------+--------------------+------------------+--------+---------+
| Header |      plain-data    |   encrypted-data  |  hash  | random  |
+--------+--------------------+------------------+--------+---------+
/--------/---- plainLen -------/-- encryptedLen ---/-- 32 --/- 16 -/
/----------------- toBeHashedLen  -----------------/
                              /---------- toBeEncryptedLen ----------/
```

DATA INTEGRITY

A secure object contains a message digest (hash, random) that ensures data integrity of the user data. The hash value is computed and stored during the wrap() operation as well as adding a random number (before data encryption takes place) and recomputed and compared during the unwrap() operation (after the data has been decrypted).

CONFIDENTIALITY

Secure objects are encrypted using context-specific keys that are never exposed, neither to the normal world, nor to the Trusted Application. It is up to the user to define how many bytes of the user data are to be kept in plain text and how many bytes are to be encrypted. The plain text part of a secure object always precedes the encrypted part.

AUTHENTICITY

As a means of ensuring the trusted origin of secure objects, the wrap operation stores the Trusted Application id (SPID, UUID) of the calling Trusted Application in the secure object header (as producer).

This allows Trusted Applications to only accept secure objects from certain partners. This is most important for scenarios involving secure object sharing.

CONTEXT

The concept of context allows for sharing of secure objects. At present there are three kinds of context:

- MC_SO_CONTEXT_TLT: Trusted Application context. The secure object is confined to a particular Trusted Application. This is the standard use case.
- PRIVATE WRAPPING: If no consumer was specified, only the Trusted Application that wrapped the secure object can unwrap it.
- DELEGATED WRAPPING: If a consumer Trusted Application is specified, only the Trusted Application specified as 'consumer' during the wrap operation can unwrap the secure object. Note that there is no delegated wrapping with any other contexts.
- MC_SO_CONTEXT_SP: Service provider context. Only Trusted Applications that belong to the same service provider can unwrap a secure object that was wrapped in the context of a certain service provider.
- MC_SO_CONTEXT_DEVICE: Device context. All Trusted Applications can unwrap secure objects wrapped for this context.

Default flag TLAPI_UNWRAP_DEFAULT permits only Trusted Application context and no delegation. Include flag TLAPI_UNWRAP_PERMIT_DELEGATED if you want to allow delegated objects. Include flags TLAPI_UNWRAP_PERMIT_CONTEXT_SP or TLAPI_UNWRAP_PERMIT_CONTEXT_DEVICE if you want to permit unwrapping with those context.

LIFETIME

The concept of a lifetime allows limiting how long a secure object is valid. After the end of the lifetime, it is impossible to unwrap the object. At present, three lifetimes are defined:

MC_SO_LIFETIME_PERMANENT: Secure Object does not expire.

MC_SO_LIFETIME_POWERCYCLE: Secure Object expires on reboot.

MC_SO_LIFETIME_SESSION: Secure Object expires when Trusted Application session is closed. The secure object is thus confined to a particular session of a particular Trusted Application. Note that session lifetime is only allowed for private wrapping in the Trusted Application context MC_SO_CONTEXT_TLT.

3.6.1 Types

3.6.1.1 tsSource_t

Real time sources in <t-base.

Enumerator:

- TS_SOURCE_ILLEGAL Illegal counter source value.
- TS_SOURCE_SOFTCNT monotonic counter that is reset upon power cycle.
- TS_SOURCE_SECURE_RTC Secure real time clock that uses underlying hardware clock.

3.6.2 Functions

3.6.2.1 tlApiUnwrapObjectExt

```
_TLAPI_EXTERN_C tlApiResult_t tlApiUnwrapObjectExt (
   void * src,
   size_t srcLen,
   void * dest,
   size_t * destLen,
   uint32_t flags)
```

Unwraps a secure object.

Decrypts and verifies the checksum of given object for the context indicated in the secure object's header.

Verifies and decrypts a secure object and stores the user data (plain data and the decrypted data) to a given location. For further details refer to tlApiWrapObject().

After this operation, the source address contains the decrypted secure object (whose user data starts immediately after the secure object header), or the attempt of the decryption, which might be garbage, in case the decryption failed (due to a wrong context, for instance).

If dest is not NULL, copies the decrypted user data part to the specified location, which may overlap with the memory occupied by the original secure object.

Parameters:

- in, out src: [in] Encrypted secure object, [out] decrypted secure object i.e. the secure object header data the plain data and the decrypted data (which was earlier encrypted by the wrapper function).
- in srcLen: Length of source buffer i.e. the length of the secure object.
- in, out dest: Address of user data or NULL if no extraction of user data is desired. Note that this buffer has to be statically allocated (which is the reason why it also is set as input parameter). The tlApiWrapObjectExt does not allocate the buffer, it only writes to the buffer from the starting address and maximum of destLen (see parameter below).
- in, out destLen: [in] Length of destination buffer. [out] Length of user data. The length of the statically allocated buffer is sent as input for copying the userdata after the decryption of the secure object. The length of the userdata is returned.
- in flags: See more explanation at the top, in the CONTEXT part.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_INVALID_INPUT if an input parameter is invalid.
- E_TLAPI_CR_WRONG_OUPUT_SIZE if the output buffer is too small.
- E_TLAPI_SO_WRONG_VERSION if the version of the secure object is not supported.
- E_TLAPI_SO_WRONG_TYPE if secure object type is not supported.
- E_TLAPI_SO_WRONG_LIFETIME if the kind of lifetime of the secure object is not supported.
- E_TLAPI_SO_WRONG_CONTEXT if the kind of context of the secure object is not supported.
- E_TLAPI_SO_WRONG_CHECKSUM if (after decryption) the checksum over the whole secure object (header and payload) is wrong. This is usually an indication that the secure object has been tampered with, or that the client calling unwrap is not allowed to unwrap the secure object.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.
- E_TLAPI_SO_DELEGATED_NOT_PERMITTED Delegated objects were not permitted.
- E_TLAPI_SO_CONTEXT_NOT_PERMITTED This context was not permitted.

3.6.2.2 tlApiWrapObjectExt

```
_TLAPI_EXTERN_C tlApiResult_t tlApiWrapObjectExt (
      const void * src,
      size_t plainLen, 3
      size_t encryptedLen,
      void * dest,
      size_t * destLen,
      mcSoContext_t context,
      mcSoLifeTime_t lifetime,
      const tlApiSpTrustletId_t * consumer,
      uint32_t flags)
```

Wraps user data given in the source buffer and creates a secure object in the destination buffer.

The required size of the destination buffer (total size of secure object) can be obtained through the MC_SO_SIZE() macro. The input to this macro is the length of the plain data and the length of the data that is to be encrypted.

Example:

```
  secureObjectLength = MC_SO_SIZE(plainLength, encryptLength)
```

Since dynamic memory allocations are not supported in the Secure World, i.e. the Trusted Applications must allocate memory statically, the MC_SO_SIZE() macro can be used for statically allocating memory with the size of the Secure Object. Example:

```
  uint8_t dataBuf[MC_SO_SIZE(plainLength, encryptLength)];

Source and destination addresses may overlap, thus the following code
is a typical usage pattern:

  union {      uint8_t src[100];      uint8_t dst[MC_SO_SIZE(30, 70)];
// 30 bytes plain, 70 bytes encrypted.  } buffer;


  size_t soLen = sizeof(buffer);

  // Fill source.  buffer.src[0] = 'H';  ...

  if  (TLAPI_OK  ==  tlApiWrapObject(            buffer.src,  30,  70,
buffer.dst, &soLen, MC_SO_CONTEXT_TLT, NULL)) {      ...  }
```

Parameters:

- in src: User data. The data which is created by the user. The user data is divided into two types i.e. data that will remain cleartext and will not be encrypted and the data that will be encrypted into the secure object. Note! It can be a good programming exercise/experiment to check the Secure Object data and there find out that some part of the SO is plain-text and therefore readable.
- In plainLen: Length of plain text user data (from beginning of src). This is the length of the userdata that is going to remain as plain text (plain data), i.e. not be encrypted.
- in encryptedLen: Length of to-be-encrypted user data (after plain text user data). This is the Length of the data that is going to be encrypted. The offset is after the last byte of the plain text.

- in, out dest: Destination buffer (secure object). Every secure object starts with the header, so pass the header into this function. Note that the pointer must be word-aligned.
- in, out destLen: [in] Length of the destination buffer. [out] Length of the secure object.
- in context: Key context.
- in lifetime: Expiry type of secure object.
  - MC_SO_LIFETIME_PERMANENT: Secure Object does not expire.
  - MC_SO_LIFETIME_POWERCYCLE: Secure Object expires on reboot.
  - MC_SO_LIFETIME_SESSION: Secure Object expires when Trusted Application session is closed.
- in consumer: NULL or Trusted Application/service provider identifier for delegated wrapping. Delegated wrapping makes it possible for other Trusted Applications to unwrap the secure object. Such scenario can be communication between trustlets. It can be a service provider that is using several trustlets which are communicating with each other.
- in flags: Use the TLAPI_WRAP_DEFAULT flag

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER If a pointer input parameter is NULL.
- E_TLAPI_INVALID_INPUT If an input parameter is invalid, for instance if the maximum payload size is exceeded.
- E_TLAPI_CR_WRONG_OUPUT_SIZE If the output buffer is too small.
- E_TLAPI_UNALIGNED_POINTER If the secure object pointer is not word-aligned. E_TLAPI_UNMAPPED_BUFFER If one buffer is not entirely mapped in Trusted Application address space.

3.7 SYSTEM FUNCTIONS

The <t-base system API interface provides system information and system functions to Trusted Applications.

3.7.1 Functions

3.7.1.1 tlApiExit

```
_TLAPI_EXTERN_C _TLAPI_NORETURN void tlApiExit (
    uint32_t exitCode)
```

Returns:

TLAPI_OK if character c has successfully been read.

E_TLAPI_BUFFER_TOO_SMALL if mcPlatformInfo.size is too small. On return mcPlatformInfo.size will be set to the required length. Terminate the Trusted Application with an exit code. Trusted Applications can use the **tlApiExit()** to terminate themselves and return an exit code. This can be used if the initialization fails or an unrecoverable error occurred. The Trusted Application will be terminated immediately and this function will not return.

Parameters:

- exitCode : exit code

3.7.1.2 tlApiGetPlatformInformation

```
_TLAPI_EXTERN_C _TLAPI_NORETURN void tlApiGetPlatformInformation (
      tlApiPlatformInfo_t platformInfo)
```

Get information about the hardware platform.

The function tlApiGetPlatformInformation() provides information about the current hardware platform.

Parameters:

- platformInfo: pointer to tlApiPlatformInfo_t structure that receives the platform information.

Returns:

There is no return code, since the function will not return.

3.7.1.3 tlApiGetMobicoreVersion

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetMobicoreVersion (
     mcVersionInfo_t * mcVersionInfo)
```

Get information about the underlying <t-base version.

The function **tlApiGetMobicoreVersion()** provides the <t-base product id and version information about the various <t-base interfaces as defined in mcVersionInfo.h

Parameters:

- mcVersionInfo: pointer to version information structure.

Returns:

- TLAPI_OK if version has been set
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.7.1.4 tlApiGetSuid

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetSuid (
    mcSuid_t * suid)
```

Get the System on Chip Universal Identifier.

Parameters:

- suid: pointer to Suid structure that receives the Suid data

Returns:

- TLAPI_OK if Suid has been successfully read.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.7.1.5 tlApiGetVirtMemType

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetVirtMemType(
        uint32_t *type,
        addr_t   addr,
        uint32_t size);
```

Get the virtual memory type

Parameters:

- in *type: pointer to address where type is returned
- in addr: start address of checked memory
- in size: size checked memory

Returns:

- TLAPI_VIRT_MEM_TYPE_SECURE The memory area is mapped as secure
- TLAPI_VIRT_MEM_TYPE_NON_SECURE The memory area is mapped as non-secure

3.7.1.6 tlApiIsNwdBufferValid

```
bool tlApiIsNwdBufferValid( addr_t addr, uint32_t size )
```

Helper to simplify NWd buffer testing.

Parameters:

- in addr: pointer to NWd buffer.
- In size: size of NWd buffer.

Returns:

- TLAPI_OK if buffer is valid.

3.7.1.7 tlApiGetVersion

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetVersion (
    uint32_t * tlApiVersion)
```

Gets information about the implementation of the <t-base Trusted Application API version.

Parameters:

- tlApiVersion: pointer to tlApi version.

Returns:

- TLAPI_OK if version has been set
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

3.7.1.8 tlApiLogvPrintf

```
_TLAPI_EXTERN_C void tlApiLogvPrintf (
      const char * fmt,
      va_list args)
```

Formatted logging functions.

tlApiLogvPrintf, tlApiLogPrintf

Minimal printf-like function to print logging message to NWd log.

Supported formatters:

- %s String, NULL value emit "<NULL>".
- %x %X hex
- %p pointer (hex with fixed width of 8)
- %d i signed decimal
- %u unsigned decimal
- %t timestamp (if available in platform). NOTE: This does not consume any value in parameter list.
- %% outputs single %
- %s, %x, %d, and %u support width (example %5s). Width is interpreted as minimum number of characters. Hex number is left padded using '0' to desired width. Decimal number is left padded using ' ' to desired width. String is right padded to desired length.

Newline is used to terminate logging line.

Parameters:

- fmt: Formatter

3.8 TRUSTED USER INTERFACE

3.8.1 Header File

The header file for the Trusted User Interface (TUI) API is "`TlApiTui.h`".

```
#include "TlApiTui.h"
```

3.8.2 Error Codes

Constant Name	Value	Definition
E_TLAPI_TUI_NO_SESSION	0x00000501	The session to TUI driver cannot be found. It was not opened or has been closed.
E_TLAPI_TUI_BUSY	0x00000502	TUI driver is busy. Another session may be open.
E_TLAPI_TUI_NO_EVENT	0x00000503	No TUI event has occurred since the session started or the last call of get event.
E_TLAPI_TUI_OUT_OF_DISPLAY	0x00000504	The coordinates/size of a displayable object are at least partially out of the of display area.

Table 12: Trusted Application API Error Codes

3.8.3 Types

3.8.3.1 tlApiTuiScreenInfo_t

```
typedef struct {
    uint32_t    grayscaleBitDepth;
    uint32_t    redBitDepth;
    uint32_t    greenBitDepth;
    uint32_t    blueBitDepth;
    uint32_t    width;
    uint32_t    height;
    uint32_t    wDensity;
    uint32_t    hDensity;
} tlApiTuiScreenInfo_t, *tlApiTuiScreenInfo_ptr;
```

General information about the screen.

The fields of the structure are:

- grayscaleBitDepth: Available grayscale depth.
- redBitDepth: Available red bit depth.
- greenBitDepth: Available green bit depth.
- blueBitDepth: Available blue bit depth.
- width: Width of the screen in pixel.
- height: Height of the screen in pixel.
- wDensity: Density of the screen in pixel-per-inch.
- hDensity: Density of the screen in pixel-per-inch.

3.8.3.2 tlApiTuiTouchEventType_t

Type of touch event.

Enumerator:

TUI_TOUCH_EVENT_RELEASED: A pressed gesture has finished.
TUI_TOUCH_EVENT_PRESSED: A pressed gesture has occurred.

3.8.3.3 tlApiTuiImage_t

```
typedef struct {
    void*       imageFile;
    uint32_t    imageFileLength;
} tlApiTuiImage_t, *tlApiTuiImage_ptr;
```

Image file.

The fields of the structure are:

- imageFile: a buffer containing the image file.
- imageFileLength: size of the buffer.

3.8.3.4 tlApiTuiCoordinates_t

```
typedef struct {
    uint32_t    xOffset;
    uint32_t    yOffset;
} tlApiTuiCoordinates_t, *tlApiTuiCoordinates_ptr;
```

Coordinates.

These are related to the top-left corner of the screen.

The fields of the structure are:

- xOffset: x coordinate.
- xOffset: y coordinate.

3.8.3.5 tlApiTuiTouchEvent_t

```
typedef struct {
    tlApiTuiTouchEventType_t   type;
    tlApiTuiCoordinates_t      coordinates;
} tlApiTuiTouchEvent_t, *tlApiTuiTouchEvent_ptr;
```

Touch event data.

The fields of the structure are:

- type: type of touch event.
- coordinates: coordinates of the touch event in the screen.

3.8.4 Functions

3.8.4.1 tlApiTuiGetScreenInfo

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetScreenInfo (
    tlApiTuiScreenInfo_ptr screenInfo)
```

Get screen information.

Parameters:

- screenInfo: screen information.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.

3.8.4.2 tlApiTuiOpenSession

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiOpenSession (void)
```

Open a session to the TUI driver.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_TUI_BUSY if the TUI driver cannot be opened.

3.8.4.3 tlApiTuiCloseSession

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiCloseSession (void)
```

Close the session to the TUI driver.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_TUI_NO_SESSION if the TUI driver session cannot be found. It was not opened or has been closed.

3.8.4.4 tlApiTuiSetImage

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiSetImage (
      tlApiTuiImage_ptr image,
      tlApiTuiCoordinates_t coordinates)
```

Draw an image in secure display.

Parameters:

- image: image to be displayed.
- coordinates: coordinates where to display the image in the screen, related to the top left corner.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_TUI_NO_SESSION if the TUI driver session cannot be found. It was not opened or has been closed.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_DRV_INVALID_PARAMETERS if one parameter is not valid.
- E_TLAPI_NOT_IMPLEMENTED if the format of the image file is not supported.
- E_TLAPI_TUI_OUT_OF_DISPLAY if the image or a part of the image is out of the display area.

3.8.4.5 tlApiTuiGetTouchEvent

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetTouchEvent (
    tlApiTuiTouchEvent_ptr touchEvent)
```

Get a touch event from TUI driver.

This is non-blocking call. It shall be called when the TL is notified.

Parameters:

- touchEvent: the touch event that occurred.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_TUI_NO_SESSION if the TUI driver session cannot be found. It was not opened or has been closed.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.

3.9 DRM API

3.9.1 Structures

3.9.1.1 tlApiDrmOffsetSizePair

```
typedef struct
{
    uint32_t nSize;              /* Size of encrypted region */
    uint32_t nOffset;            /* offset to encrypted region */
} tlApiDrmOffsetSizePair_t;
```

Structure containing the offset and size of an encrypted data section within a buffer, potentially one of many sections within the buffer.

3.9.1.2 tlApiDrmAlg

```
typedef enum {
    DRV_NONE,
    DRV_AES_ECB,
    DRV_AES_CBC,
    DRV_AES_CTR32,
    DRV_AES_CTR64,
    DRV_AES_CTR96,
    DRV_AES_CTR128,
    DRV_AES_XTS,
    DRV_AES_CBCCTS
} tlApiDrmAlg_t;
```

Enum continaing list of cryptographic algorithms available.

3.9.1.3 tlApiDrmLink

```
typedef enum {
    HDCP_1,
    HDCP_2,
    AIRPLAY,
    DTCP
} tlApiDrmLink_t;
```

Structure containing the type of output link that needs to be protected and checked according to license.

3.9.1.4 tlApiDrmInputSegmentDescriptor

```
typedef struct
{
 uint32_t nTotalSize;        /** size of buffer (plain + encrypted) */
 uint32_t nNumBlocks;                  /* No. of encrypted regions */
 tlApiDrmOffsetSizePair_t aPairs[10]; /* Array of offset/size pairs */
}tlApiDrmInputSegmentDescriptor;
```

Structure containing the number of encrypted regions in the buffer and their offset/size information.

3.9.1.5 tlApiDrmDecryptContext

The crypto context to contain all IV, key and algorithm information required to decrypt the content.

```
/**
 * For DRM cipher/copy operations
 *
 * Parameters
 * @param  key          [in]  content key
 * @param  key_len      [in]  key length in bytes (16,24,32)
 * @param  iv           [in]  initialization vector. Always 16 bytes.
 * @param  ivlen        [in]  length initialization vector.
 * @param  alg          [in]  algorithm
 * @param  outputoffset [in]  output data offset
 *
 */
typedef struct tlApiDrmDecryptContext
{
    uint8_t                 *key;
    int32_t                 keylen;
    uint8_t                 *iv;
    uint32_t                ivlen;
    TL_DRM_Alg_t            alg;
    uint32_t                outputoffet;
}tlApiDrmDecryptContext;
```

3.9.2 Constants

Constant Name	Value	Definition
TL_DRM_KEY_SIZE_128	16	Key size supported for AES Cipher.
TL_DRM_KEY_SIZE_192	24	Key size supported for AES Cipher.
TL_DRM_KEY_SIZE_256	32	Key size supported for AES Cipher.
TL_DRM_PROCESS_ENCRYPTED_DATA	1	Indicates encrypted data is being passed to the driver.
TL_DRM_PROCESS_DECRYPTED_DATA	2	Indicates decrypted data is being passed to the driver.

Table 13 Constants

3.9.3 Errors

Constant Name	Value	Definition
E_TLAPI_DRM_OK	0	No Error.
E_TLAPI_DRM_INVALID_PARAMS	0x601	Invalid parameter for Cipher
E_TLAPI_DRM_INTERNAL	0x602	Internal error in AES
E_TLAPI_DRM_MAP	0x603	Driver mapping error
E_TLAPI_DRM_PERMISSION_DENIED	0x604	Permission Denied
E_TLAPI_DRM_REGION_NOT_SECURE	0x605	If the output address is not protected.

E_TLAPI_DRM_SESSION_NOT_AVAILABLE	0x606	If a single session implementation is already active, or a multi session implementation has no free sessions.
E_TLAPI_DRM_INVALID_COMMAND	0x607	If the command ID received is unrecognized.
E_TLAPI_DRM_ALGORITHM_NOT_SUPPORTED	0x608	If the requested algorithm is not supported by the driver. If this error is thrown the trusted application must decipher the content itself.
E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED	0x609	If the functions have not been implemented.

Table 14 Secure Playback Errors

3.9.4 Functions

3.9.4.1 tlApiDrmOpenSession

```
tlApiResult_t tlApiDrmOpenSession(
      int *sHandler);
```

If multiple session support is required it must be first managed here, this function is also required to set up any initial requirements in the hardware prior to decryption for example (if required according to platform and chose framework architecture) :

- Initialize the Crypto Hardware
- Initialize the Media Framework
- Authenticate the decoder firmware.
- Enable Firewalls.

Parameters:

sHandle: [out] Session Handle of the new TA session.

Returns:

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INTERNAL` general error in case of crypto problem
- `E_TLAPI_DRM_MAP` in case of error mapping memory to driver.
- `E_TLAPI_DRM_PERMISSION_DENIED` in case of rights access related issue
- `E_TLAPI_DRM_SESSION_NOT_AVAILABLE` in case the driver is busy and cannot open a session.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

3.9.4.2 tlApiDrmProcessContent

```
tlApiResult_t tlApiDrmProcessContent (
    uint8_t                             sHandle,
    tlApiDrmDecryptContext              decryptCtx,
    uint8_t                             *input,
    tlApiDrmInputSegmentDescriptor      inputDesc,
    uint16_t                            processMode,
    uint8_t                             *rfu);
```

Processes the specified content.

If the algorithm is supported by the driver this function is used to decrypt the encrypted data into a protected buffer. If the algorithm is not supported it will respond in an error. In this case the decryption should be done by the Trusted Application and the decrypted data shall be copied to the protected buffer using this function with `processMode` set to `TL_DRM_PROCESS_DECRYPTED_DATA`.

The parameter `processMode` is a constant value indicating whether encrypted or decrypted data is being treated from the TA.

If multiple sessions are supported, the `sHandle` parameter is used to identify the session requested for decryption.

Input, key and iv data provided within the TL_DRM_DecryptContext structure indicates the context of the cryptographic operation.

If a frame consists of multiple encrypted areas, the TL_DRM_InputSegementDescriptor structure must hold the offsets and lengths of the encrypted regions, the offsets will also correspond to the offsets in the output buffer. If the input is merely one encrypted buffer, this will be indicated by the structure. If the input buffer to the TA contains both clear and encrypted data then both clear and encrypted data must be passed to the driver using this function.

If `processmode` is `TL_DRM_PROCESS_DECRYPTED_DATA` the structure elements: key, keylen and iv are ignored as they are irrelevant for the copy.

Parameters:

- sHandle: [in] Session Handle of the current TA session.
- decryptCtx: [in] Contains the IV, Key, Key length and all necessary crypto information for the requested algorithm.
- input: [in] Address to the start of a block of encrypted data, or if required multiple sections of encrypted and decrypted segments the offsets and legths of which are described in the next parameter.
- inputDesc: [in] Structure containing offsets and lengths of data to be decrypted if multiple segements are present within the same buffer, if not it will contain the offset and length of the only encrypted segment.
- processMode: [in] states whether the incoming data is decrypted or encrypted, which infers a secure copy, or a decrypt operation is required.
- rfu: [rfu] Reserved for future use.

Returns:

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INVALID_PARAMS` incorrect parameters in input.
- `E_TLAPI_DRM_INTERNAL` general Error in case of crypto problem
- `E_TLAPI_DRM_MAP` in case of error mapping memory to driver.
- `E_TLAPI_DRM_PERMISSION_DENIED` in case of rights access related issue

- `E_TLAPI_DRM_REGION_NOT_SECURE` if the memory for output is not protected
- `E_TLAPI_DRM_ALGORITHM_NOT_SUPPORTED` in case the algorithm is not supported.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

3.9.4.3 tlApiDrmCloseSession

```
tlApiResult_t tlApiDrmCloseSession(
      int sHandler);
```

Closes the DRM session.

It operates on the session indicated by the session handle passed in the function. If multi session is not supported, this value is ignored.

Parameters:

sHandle: [in] Session Handle of the current TA session.

Returns:

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INTERNAL` in case of failure.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

3.9.4.4 tlApiDrmCheckLink

```
tlApiResult_t tlApiDrmCheckLink(
      uint8_t sHandler,
      tlApiDrmLink_t link)
```

This function is used to check the protected external link information like HDCPv1, HDCPv2, AirPlay and DTCP.

It operates on the session indicated by the session handle passed in the function. If multi-session is not supported, this value is ignored.

Parameters:

sHandle: [in] Session Handle of the current TA session.

link: [in] external link information like HDCPv1, HDCPv2, AirPlay, and DTCP.

Returns:

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INTERNAL` in case of failure.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

3.10 MEMORY MANAGEMENT

3.10.1 Header File

The header file for the Memory Management functions is "`TlApiHeap.h`".

```
#include "TlApiHeap.h"
```

3.10.2 Functions

3.10.2.1 tlApiMalloc

```
void* tlApiMalloc(uint32_t size, uint32_t hint);
```

Allocates a block of memory from the heap.

The address of the allocated block is aligned on a 8-bytes boundary. A block allocated by `tlApiMalloc` must be freed by `tlApiFree`.

If the size of the space requested is zero, the value returned is still a non-NULL pointer that the Trusted Application must not attempt to access.

Parameters:

- `size`: [in] the number of bytes to be allocated.
- `hint`: [in] must be 0

Returns:

- Upon successful completion, with size not equal to zero, the function returns a pointer to the allocated space. Otherwise, a NULL pointer is returned.

3.10.2.2 tlApiRealloc

```
void* tlApiRealloc(void* buffer, uint32_t newSize);
```

Reallocates a block of memory from a heap.

This function allows resizing a memory block.

If buffer is NULL, tlApiRealloc is equivalent to tlApiMalloc.

If buffer is not NULL and newSize is 0, then tlApiRealloc is equivalent to tlApiFree and returns a non-NULL pointer that the Trusted Application must not attempt to access.

If newSize is less or equal to the current size of the block, the block is truncated, the content of the block is left unchanged and the function returns buffer.

If newSize is greater than the current size of the block, the size of the block is increased. The whole content of the block is copied at the beginning of the new block. If possible, the block is enlarged in place and the function returns buffer. If this is not possible, a new block is allocated with the new size, the content of the current block is copied, the current block is freed and the function returns the pointer on the new block.

Parameters:

- buffer: [in] Pointer to the block of memory that the function reallocates. This value may be null or returned by an earlier call to tlApiMalloc or tlApiRealloc.
- newSize: [in] size of the memory block in bytes. This value may be zero.

Returns:

- A pointer to the reallocated memory block, a non-NULL pointer if the newSize is zero or NULL if an error is detected.

3.10.2.3 tlApiFree

```
void tlApiFree(void* buffer);
```

Frees a memory block allocated from a heap by tlApiMalloc or tlApiRealloc.

This function does nothing if buffer is NULL.

Parameters:

- `buffer`: [in] Pointer to the block of memory to be freed.

4 mcClient API

The mcClient API is the Legacy API for developing Client Applications.

Note that the same API is also available at the kernel-level for kernel modules.

4.1 HEADER FILE

The header file for the mcClient API is "`MobicoreDriverApi.h`".

```
#include "MobicoreDriverApi.h"
```

4.2 DATA STRUCTURES

4.2.1 mcBulkMap_t Structure Reference

```
typedef struct {
    void *sVirtualAddr;
    uint32_t sVirtualLen;
} mcBulkMap_t;
```

Information structure about additional mapped Bulk buffer between the Trusted Application Connector (Nwd) and the Trusted Application (Swd). This structure is initialized from a Trusted Application Connector by calling mcMap(). In order to use the memory within a Trusted Application the Trusted Application Connector has to inform the Trusted Application about the content of this structure via the TCI.

The fields of the structure are:

- sVirtualAddr: The virtual address of the Bulk buffer regarding the address space of the Trusted Application, already includes a possible offset!
- sVirtualLen: Length of the mapped Bulk buffer

4.2.2 mcSessionHandle_t Structure Reference

Structure of Session Handle, includes the Session ID and the Device ID the Session belongs to. The session handle will be used for session-based <t-base communication.

It will be passed to calls which address a communication end point in the <t-base environment.

```
typedef struct {
    uint32_t sessionId;
    uint32_t deviceId;
} mcSessionHandle_t;
```

The fields of the structure are:

- sessionId: session ID
- deviceId: Device ID the session belongs to

4.1 ERROR CODES

Macro Name	Definition
MC_DRV_ERROR_MAJOR(ecode)	Get MAJOR part of error code.
MC_DRV_ERROR_MCP(ecode)	Get MCP part of error code.
MC_DRV_ERROR_DETAIL(ecode)	Get detail part of error code.

Table 15 Trusted Application API Error Codes

Constant Name	Definition
MC_DRV_OK	Function call succeeded.
MC_DRV_NO_NOTIFICATION	No notification available.
MC_DRV_ERR_NOTIFICATION	Error during notification on communication level.
MC_DRV_ERR_NOT_IMPLEMENTED	Function not implemented.
MC_DRV_ERR_OUT_OF_RESOURCES	No more resources available.
MC_DRV_ERR_INIT	Driver initialization failed.
MC_DRV_ERR_UNKNOWN	Unknown error.
MC_DRV_ERR_UNKNOWN_DEVICE	The specified device is unknown.
MC_DRV_ERR_UNKNOWN_SESSION	The specified session is unknown.
MC_DRV_ERR_INVALID_OPERATION	The specified operation is not allowed.
MC_DRV_ERR_INVALID_RESPONSE	The response header from the MC is invalid.
MC_DRV_ERR_TIMEOUT	Function call timed out.
MC_DRV_ERR_NO_FREE_MEMORY	Cannot allocate additional memory.
MC_DRV_ERR_FREE_MEMORY_FAILED	Free memory failed.
MC_DRV_ERR_SESSION_PENDING	Still some open sessions pending.
MC_DRV_ERR_DAEMON_UNREACHABLE	MC daemon not reachable
MC_DRV_ERR_INVALID_DEVICE_FILE	The device file of the kernel module could not be opened.
MC_DRV_ERR_INVALID_PARAMETER	Invalid parameter.

MC_DRV_ERR_KERNEL_MODULE	Error from Kernel Module, see DETAIL for more.
MC_DRV_ERR_BULK_MAPPING	Error during mapping of additional bulk memory to session.
MC_DRV_ERR_BULK_UNMAPPING	Error during un-mapping of additional bulk memory to session.
MC_DRV_INFO_NOTIFICATION	Notification received, exit code available.
MC_DRV_ERR_NQ_FAILED	Setup of NWd connection failed.
MC_DRV_OK	Function call succeeded.
MC_DRV_NO_NOTIFICATION	No notification available.
MC_DRV_ERR_NOTIFICATION	Error during notification on communication level.
MC_DRV_ERR_NOT_IMPLEMENTED	Function not implemented.
MC_DRV_ERR_OUT_OF_RESOURCES	No more resources available.
MC_DRV_ERR_INIT	Driver initialization failed.
MC_DRV_ERR_UNKNOWN	Unknown error.
MC_DRV_ERR_UNKNOWN_DEVICE	The specified device is unknown.
MC_DRV_ERR_UNKNOWN_SESSION	The specified session is unknown.
MC_DRV_ERR_INVALID_OPERATION	The specified operation is not allowed.
MC_DRV_ERR_INVALID_RESPONSE	The response header from the MC is invalid.
MC_DRV_ERR_TIMEOUT	Function call timed out.
MC_DRV_ERR_NO_FREE_MEMORY	Cannot allocate additional memory.
MC_DRV_ERR_FREE_MEMORY_FAILED	Free memory failed.
MC_DRV_ERR_DAEMON_UNREACHABLE	MC daemon not reachable.
MC_DRV_ERR_INVALID_DEVICE_FILE	The device file of the kernel module could not be opened.
MC_DRV_ERR_INVALID_PARAMETER	Invalid parameter.
MC_DRV_ERR_BULK_MAPPING	Error during mapping of additional bulk memory to session.
MC_DRV_ERR_BULK_UNMAPPING	Error during un-mapping of additional bulk memory to session.
MC_DRV_INFO_NOTIFICATION	Notification received, exit code available.
MC_DRV_ERR_NQ_FAILED	Set up of NWd connection failed.

MC_DRV_ERR_DAEMON_VERSION	Wrong daemon version.
MC_DRV_ERR_CONTAINER_VERSION	Wrong container version.
MC_DRV_ERR_WRONG_PUBLIC_KEY	System Trusted Application public key is wrong.
MC_DRV_ERR_CONTAINER_TYPE_MISMATCH	Wrong container type(s).
MC_DRV_ERR_CONTAINER_LOCKED	Container is locked (or not activated).
MC_DRV_ERR_SP_NO_CHILD	SPID is not registered with root container.
MC_DRV_ERR_TL_NO_CHILD	UUID is not registered with SP container.
MC_DRV_ERR_UNWRAP_ROOT_FAILED	Unwrapping of root container failed.
MC_DRV_ERR_UNWRAP_SP_FAILED	Unwrapping of service provider container failed.
MC_DRV_ERR_UNWRAP_TRUSTLET_FAILED	Unwrapping of Trusted Application container failed.
MC_DRV_ERR_DEVICE_ALREADY_OPEN	Device is already open.
MC_DRV_ERR_SOCKET_CONNECT	MC daemon socket not reachable.
MC_DRV_ERR_SOCKET_WRITE	MC daemon socket write error.
MC_DRV_ERR_SOCKET_READ	MC daemon socket read error.
MC_DRV_ERR_SOCKET_LENGTH	MC daemon socket read error.
MC_DRV_ERR_DAEMON_SOCKET	MC daemon had problems with socket.
MC_DRV_ERR_DEVICE_FILE_OPEN	The device file of the kernel module could not be opened.
MC_DRV_ERR_NULL_POINTER	Null pointer passed as parameter.
MC_DRV_ERR_TCI_TOO_BIG	Requested TCI length is too high.
MC_DRV_ERR_WSM_NOT_FOUND	Requested TCI was not allocated with mallocWsm().
MC_DRV_ERR_TCI_GREATER_THAN_WSM	Requested TCI length is bigger than allocated WSM.
MC_DRV_ERR_TRUSTLET_NOT_FOUND	Trusted Application could not be found in mcRegistry.
MC_DRV_ERR_DAEMON_KMOD_ERROR	Daemon cannot use Kernel module as expected.
MC_DRV_ERR_DAEMON_MCI_ERROR	Daemon cannot use MCI as expected.
MC_DRV_ERR_MCP_ERROR	Control Protocol error. See MC_DRV_ERROR_MCP().
MC_DRV_ERR_INVALID_LENGTH	Invalid length.
MC_DRV_ERR_KMOD_NOT_OPEN	Device not open.

MC_DRV_ERR_BUFFER_ALREADY_MAPPED	Buffer is already mapped to this Trusted Application.
MC_DRV_ERR_BLK_BUFF_NOT_FOUND	Unable to find internal handle for buffer.
MC_DRV_ERR_DAEMON_DEVICE_NOT_OPEN	No device associated with connection.
MC_DRV_ERR_DAEMON_WSM_HANDLE_NOT_FOUND	Daemon could not find wsm.h
MC_DRV_ERR_DAEMON_UNKNOWN_SESSION	The specified session is unknown by the daemon
MAKE_MC_DRV_MCP_ERROR(mcpCode)	Macro used to build a MCP Error Code
MAKE_MC_DRV_KMOD_WITH_ERRNO(theErrno)	Macro used to build a Kernel Module Error Code
MC_DEVICE_ID_DEFAULT	The default device ID
MC_INFINITE_TIMEOUT	Wait infinite for a response of the MC.
MC_NO_TIMEOUT	Do not wait for a response of the MC.
MC_MAX_TCI_LEN	TCI/DCI must not exceed 1MiB

Table 16 mcClient API Constants

4.2 FUNCTIONS

4.2.1 mcOpenDevice

```
__MC_CLIENT_LIB_API mcResult_t mcOpenDevice (uint32_t deviceId)
```

Initializes all device specific resources required to communicate with an <t-base instance located on the specified device in the system. If the device does not exist the function will return `MC_DRV_ERR_UNKNOWN_DEVICE`.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in deviceId: Identifier for the <t-base device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_ERR_INVALID_OPERATION if device already opened.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_UNKNOWN_DEVICE when device Id is unknown.

MC_DRV_ERR_INVALID_DEVICE_FILE if kernel module underdev/mobicore cannot be opened

4.2.2 mcCloseDevice

```
__MC_CLIENT_LIB_API mcResult_t mcCloseDevice (uint32_t deviceId)
```

Close the connection to a <t-base device.

When closing a device, active sessions have to be closed beforehand. Resources associated with the device will be released. The device may be opened again after it has been closed.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in deviceId: Identifier for the <t-base device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_ERR_UNKNOWN_DEVICE when device Id is unknown.
- MC_DRV_ERR_SESSION_PENDING when a session is still open.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.

4.2.3 mcOpenTrustlet

```
__MC_CLIENT_LIB_API mcResult_t mcOpenTrustlet(
    mcSessionHandle_t*      session,
    mcSpid_t                spid,
    uint8_t*                trustlet,
    uint32_t                tLen,
    uint8_t*                tci,
    uint32_t                tciLen
);
```

Opens a new session to a Trusted Application.

`session.deviceId` must be set to the device id of a device opened with a call to `mcOpenDevice`.

The `trustlet` memory buffer must contain the Trusted Application encoded binary.

The `spid` structure must be filled to indicate the ID of the Service Provider of the Trusted Application. This parameter is ignored for System Trusted Applications.

The caller must allocate the `tci` communication buffer prior to calling this function. This buffer must not be freed until the session is closed through a call to `mcCloseSession`.

When this function returns MC_DRV_OK the `session` structure has been populated with any implementation-defined information necessary for subsequent operations within the session.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in, out `session`: Before calling the function `session.deviceId` has to be filled with the device id of a previously opened device. On success, the required fields will be filled.
- in `spid`: Service Provider ID. Ignored for System Trusted Applications
- in `trustlet` : memory buffer containing the Trusted Application binary
- in `tlen` : length of the memory buffer containing the Trusted Application
- in `tci`: Communication buffer for communicating with the Trusted Application.
- in `tciLen`: Length of the TCI buffer. Maximum allowed value is `MC_MAX_TCI_LEN`.

Return:

- `MC_DRV_OK` if operation has been successfully completed.
- `MC_DRV_INVALID_PARAMETER` if session parameter is invalid.
- `MC_DRV_ERR_UNKNOWN_DEVICE` when device id is invalid.
- `MC_DRV_ERR_DAEMON_UNREACHABLE` when problems with daemon socket occur.
- `MC_DRV_ERR_UNKNOWN_DEVICE` when daemon returns an error.

4.2.4 mcOpenSession – DEPRECATED

```
__MC_CLIENT_LIB_API mcResult_t mcOpenSession (
      mcSessionHandle_t*    session,
      const mcUuid_t*       uuid,
      uint8_t*              tci,
      uint32_t              tciLen)
```

This function is deprecated – mcOpenTrustlet should be used instead.

Open a new session to a System Trusted Application.

The Trusted Application with the given UUID has to be available in the flash filesystem.

Write MCP open message to buffer and notify <t-base about the availability of a new command. Waits till the <t-base responds with the new session ID (stored in the MCP buffer).

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in, out session: On success, the session data will be returned. Note that session.deviceId has to be the device id of an opened device.
- in uuid: UUID of the Trusted Application to be opened
- in tci: TCI buffer for communicating with the Trusted Application.
- in tciLen: Length of the TCI buffer. Maximum allowed value is MC_MAX_TCI_LEN.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon socket occur.
- MC_DRV_ERR_UNKNOWN_DEVICE when daemon returns an error.

4.2.5 mcCloseSession

```
__MC_CLIENT_LIB_API mcResult_t mcCloseSession (
    mcSessionHandle_t *session)
```

Close a Trusted Application session.

Close the specified <t-base session. The call will block until the session has been closed.

A mutex is locked during the execution to avoid concurrent accesses.

Precondition:

Device deviceId has to be opened in advance.

Parameters:

- in session: pointer to the session to be closed.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_INVALID_DEVICE_FILE when daemon cannot open Trusted Application file.

4.2.6 mcNotify

```
__MC_CLIENT_LIB_API mcResult_t mcNotify (mcSessionHandle_t *session)
```

Notifies the session end point about available message data. If the session parameter is correct, notify will always succeed. Corresponding errors can only be received by mcWaitNotification().

Precondition:

A session has to be opened in advance.

Parameters:

- in session: pointer to the session to notify.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

4.2.7 mcWaitNotification

```
__MC_CLIENT_LIB_API mcResult_t mcWaitNotification (
     mcSessionHandle_t*    session,
     int32_t               timeout)
```

Wait for a notification issued by the <t-base for a specific session. The timeout parameter specifies the number of milliseconds the call will wait for a notification. If the caller passes 0 as timeout value the call will immediately return. If timeout value is below 0 the call will block until a notification for the session has been received.

Attention:

If timeout is below 0, call will block: Caller has to trust the other side to send a notification to wake him up again.

Parameters:

- in session: pointer to the session which receives the notification.
- in Timeout: Time in milliseconds to wait (MC_NO_TIMEOUT : direct return, >0 : milliseconds, MC_INFINITE_TIMEOUT : wait infinitely)

Returns:

- MC_DRV_OK if notification is available.
- MC_DRV_ERR_TIMEOUT if no notification arrived in time.
- MC_DRV_INFO_NOTIFICATION if a problem with the session was encountered. Get more details with mcGetSessionErrorCode().
- MC_DRV_ERR_NOTIFICATION if a problem with the socket occurred.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

4.2.8 mcMallocWsm – DEPRECATED

```
__MC_CLIENT_LIB_API mcResult_t mcMallocWsm (
      uint32_t  deviceId,
      uint32_t  align,
      uint32_t  len,
      uint8_t** wsm,
      uint32_t  wsmFlags)
```

This function is deprecated. Standard malloc() and free () functions should be used instead.

Allocate a block of world shared memory (WSM).

The MC driver allocates a contiguous block of memory which can be used as WSM. This implicates that the allocated memory is aligned according to the alignment parameter.

Always returns a buffer of size WSM_SIZE aligned to 4K.

Parameters:

- in deviceId: The ID of an opened device to retrieve the WSM from.
- in align: The alignment (number of pages) of the memory block (e.g. 0x00000001 for 4kb).
- in len: Length of the block in bytes.
- out **wsm: pointer to the virtual address of the world shared memory block.
- in wsmFlags: Platform specific flags describing the memory to be allocated.

Attention:

align and wsmFlags fields are currently ignored.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_NO_FREE_MEMORY if no more contiguous memory is available in this size or for this process.

4.2.9 mcFreeWsm – DEPRECATED

```
__MC_CLIENT_LIB_API mcResult_t mcFreeWsm (
      uint32_t deviceId,
      uint8_t* wsm)
```

This function is deprecated. Standard malloc() and free () functions should be used instead.

Free a block of world shared memory (WSM).

The MC driver will free a block of world shared memory (WSM) previously allocated with mcMallocWsm(). The caller has to assure that the address handed over to the driver is a valid WSM address.

A mutex is locked during the execution to avoid concurrent access to the shared memory being freed.

Parameters:

- in deviceId: The address to which the given address belongs.
- in wsm: Address of WSM block to be freed.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_FREE_MEMORY_FAILED on failures.

4.2.10 mcMap

```
__MC_CLIENT_LIB_API mcResult_t mcMap (
      mcSessionHandle_t*    session,
      void*                 buf,
      uint32_t              len,
      mcBulkMap_t*          mapInfo)
```

Map additional bulk buffer between a Trusted Application Connector (TLC) and the Trusted Application (TL) for a session.

Memory allocated in user space of the TLC can be mapped as additional communication channel (besides TCI) to the Trusted Application. Limitation of the Trusted Application memory structure applies: only 6 chunks can be mapped with a maximum chunk size of 1 MB each.

A mutex is locked during the execution to avoid concurrent access to the shared memory being freed.

Attention:

It is up to the application layer (TLC) to inform the Trusted Application about the additional mapped bulk memory.

Parameters:

- in session: pointer to the session with information of the deviceId used with the sessionId. The given buffer is mapped to the session specified in the session-Handle.
- in buf: Virtual address of a memory portion (relative to TLC) to be shared with the Trusted Application, already includes a possible offset!
- in len: Length of the block in bytes.
- out mapInfo: Information structure about the mapped Bulk buffer between the TLC (Nwd) and the TL (Swd).

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_BULK_MAPPING when buf is already used as bulk buffer or when registering the buffer failed.

4.2.11 mcUnmap

```
__MC_CLIENT_LIB_API mcResult_t mcUnmap (
      mcSessionHandle_t*    session,
      void*                 buf,
      mcBulkMap_t*          mapInfo)
```

Remove additional mapped bulk buffer between Trusted Application Connector (TLC) and the Trusted Application (TL) for a session.

Attention:

The bulk buffer will immediately be unmapped from the session context. The application layer (TLC) must inform the TL about un-mapping of the additional bulk memory before calling mcUnmap!

Parameters:

- in session: pointer to the session with information of the deviceId and the sessionId. The given buffer is mapped to the session specified in the session-Handle.
- in buf: Virtual address of a memory portion (relative to TLC) to be shared with the Trusted Application, already includes a possible offset!
- in mapInfo: Information structure about the mapped Bulk buffer between the TLC (Nwd) and the TL (Swd).

Attention:

The clientlib currently ignores the len field in mapInfo.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_BULK_UNMAPPING when buf was not registered earlier or when unregistering failed.

4.2.12 mcGetSessionErrorCode

```
__MC_CLIENT_LIB_API mcResult_t mcGetSessionErrorCode (
    mcSessionHandle_t*    session,
    int32_t*              lastErr)
```

Get additional error information of the last error that occurred on a session.

After the request the stored error code will be deleted.

Parameters:

- in session: pointer to the session.
- out lastErr: Pointer to the last error in given session:
  - If >0 Trusted Application has terminated itself with this value,
  - If <0 Trusted Application is dead because of an error within the <t-base (e.g. Kernel exception). See also notificationPayload_t enum in MCI definition at "mcinq.h".

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

4.2.13 mcGetMobiCoreVersion

```
__MC_CLIENT_LIB_API mcResult_t mcGetMobiCoreVersion (
      uint32_t         deviceId,
      mcVersionInfo_t* versionInfo)
```

Parameters:

- in deviceId: the <t-base deviceId.
- out versionInfo: The <t-base version information.

5 GLOBALPLATFORM TEE CLIENT API

<t-base supports parts of the GlobalPlatform TEE Client API v1.0 available at:

- http://www.globalplatform.org/specificationsdevice.asp

Developers should refer to the GlobalPlatform specifications for details about the GlobalPlatform APIs and how to use these APIs.

This sections details the differences between this version of <t-base and the GlobalPlatform specification.

5.1 HEADER FILE

The header file for the TEE Client API is "`tee_client_api.h`".

```
#include "tee_client_api.h"
```

5.2 IMPLEMENTATION NOTES

5.2.1 TEEC_InitializeContext

```
TEEC_Result TEEC_InitializeContext (
      const char*      name,
      TEEC_Context*    context)
```

This function is supported. The argument `name` is ignored.

5.2.2 TEEC_FinalizeContext

```
TEEC_Result TEEC_FinalizeContext (
      TEEC_Context*    context)
```

This function is supported.
It is a programmer error to call `TEEC_FinalizeContext` while there are still open sessions. In practice, in this case the sessions will be closed only when the process dies. That's an acceptable consequence of a programmer error.
The function will cause a segmentation fault if the parameter `context` is not valid. The function implementation does nothing if `context` is NULL.

5.2.3 TEEC_RegisterSharedMemory

```
TEEC_Result TEEC_RegisterSharedMemory (
      TEEC_Context*         context,
      TEEC_SharedMemory*    sharedMem)
```

This function is supported.
<t-base uses a zero-copy shared memory system. The maximum buffer size is 1024 kB.
`TEEC_RegisterSharedMemory` essentially treats registered memory references exactly like temporary memory references.

5.2.4 TEEC_AllocateSharedMemory

```
TEEC_Result TEEC_AllocateSharedMemory (
      TEEC_Context*         context,
      TEEC_SharedMemory*    sharedMem)
```

This function is supported.
<t-base uses a zero-copy shared memory system. The maximum buffer size is 1024 kB.

5.2.5 TEEC_ReleaseSharedMemory

```
TEEC_Result TEEC_ReleaseSharedMemory (
      TEEC_SharedMemory*    sharedMem)
```

This function is supported.

5.2.6 TEEC_OpenSession

```
TEEC_Result TEEC_OpenSession (
      TEEC_Context*    context,
      TEEC_Session*    session,
      const TEEC_UUID* destination,
      uint32_t         connectionMethod,
      const void*      connectionData,
      TEEC_Operation*  operation,
      uint32_t*        returnOrigin)
```

This function is supported.
The destination UUID points to the filename of <t-base System or Service Provider Trustlet in the <t-base registry.
The `connectionMethod` parameter must be `TEEC_LOGIN_PUBLIC`, otherwise return `TEEC_ERROR_NOT_IMPLEMENTED`.
The `connectionData` parameter is ignored.

5.2.7 TEEC_CloseSession

```
TEEC_Result TEEC_CloseSession (
      TEEC_Session*    session)
```

This function is supported.

5.2.8 TEEC_InvokeCommand

```
TEEC_Result TEEC_InvokeCommand (
      TEEC_Session*    session,
      uint32_t         commandID,
      TEEC_Operation*  operation,
      uint32_t*        returnOrigin)
```

This function is supported.
If the `operation` parameter references a memory region, the respective memory will be mapped to the Trusted Application for the duration of the function call.
The maximum buffer size is 1024 kB for each parameter.

5.2.9 TEEC_RequestCancellation

```
TEEC_Result TEEC_RequestCancellation (
      TEEC_Operation*  operation)
```

This function is supported.

6 GLOBALPLATFORM TEE INTERNAL API

<t-base supports parts of the GlobalPlatform TEE Internal API v1.0 available at:

- http://www.globalplatform.org/specificationsdevice.asp

Developers should refer to the GlobalPlatform specifications for details about the GlobalPlatform APIs and how to use these APIs.

This sections details the differences between this version of <t-base and the GlobalPlatform specification.

6.1 HEADER FILE

The header file for the TEE Client API is "`tee_internal_api.h`".

```
#include "tee_internal_api.h"
```

6.2 IMPLEMENTATION NOTES

The following table lists all the functions of the TEE Internal API and indicates the status for each function:

Function	Supported / Not Supported	Comment
Asymmetric		
TEE_AsymmetricDecrypt	Supported	
TEE_AsymmetricEncrypt	Supported	
TEE_AsymmetricSignDigest	Supported	
TEE_AsymmetricVerifyDigest	Supported	
Authenticated Encryption		
TEE_AEDecryptFinal	Not Supported	
TEE_AEEncryptFinal	Not Supported	
TEE_AEInit	Not Supported	
TEE_AEUpdate	Not Supported	
TEE_AEUpdateAAD	Not Supported	
Basic Arithmetic		
TEE_BigIntAdd	Not Supported	
TEE_BigIntDiv	Not Supported	
TEE_BigIntMul	Not Supported	

TEE_BigIntNeg	Not Supported	
TEE_BigIntSquare	Not Supported	
TEE_BigIntSub	Not Supported	
Cancellation		
TEE_GetCancellationFlag	Supported	
TEE_MaskCancellation	Supported	
TEE_UnmaskCancellation	Supported	
Converter		
TEE_BigIntConvertFromOctetString	Not Supported	
TEE_BigIntConvertFromS32	Not Supported	
TEE_BigIntConvertToOctetString	Not Supported	
TEE_BigIntConvertToS32	Not Supported	
Data Stream Access		
TEE_ReadObjectData	Supported	See section 6.2.5
TEE_SeekObjectData	Supported	
TEE_TruncateObjectData	Supported	See section 6.2.5
TEE_WriteObjectData	Supported	See section 6.2.5
Fast Modular Multiplication		
TEE_BigIntComputeFMM	Not Supported	
TEE_BigIntConvertFromFMM	Not Supported	
TEE_BigIntConvertToFMM	Not Supported	
Generic Object		
TEE_CloseObject	Supported	
TEE_GetObjectBufferAttribute	Supported	
TEE_GetObjectInfo	Supported	
TEE_GetObjectValueAttribute	Supported	
TEE_RestrictObjectUsage	Not Supported	
Generic Operation		
TEE_AllocateOperation	Supported	See section 6.2.6
TEE_CopyOperation	Not Supported	

TEE_FreeOperation	Supported	
TEE_GetOperationInfo	Supported	
TEE_ResetOperation	Not Supported	
TEE_SetOperationKey	Supported	
TEE_SetOperationKey2	Not Supported	
Initialization		
TEE_BigIntInit	Not Supported	
TEE_BigIntInitFMM	Not Supported	
TEE_BigIntInitFMMContext	Not Supported	
Internal Client API		
TEE_CloseTASession	Not Supported	
TEE_InvokeTACommand	Not Supported	
TEE_OpenTASession	Not Supported	
Key Derivation		
TEE_DeriveKey	Not Supported	
Logical Operation		
TEE_BigIntCmp	Not Supported	
TEE_BigIntCmpS32	Not Supported	
TEE_BigIntGetBit	Not Supported	
TEE_BigIntGetBitCount	Not Supported	
TEE_BigIntShiftRight	Not Supported	
MAC		
TEE_MACCompareFinal	Supported	
TEE_MACComputeFinal	Supported	
TEE_MACInit	Supported	
TEE_MACUpdate	Supported	
Memory Allocation and Size of Objects		
TEE_BigIntFMMContextSizeInU32	Not Supported	
TEE_BigIntFMMSizeInU32	Not Supported	
TEE_BigIntSizeInU32 (macro)	Not Supported	

Memory Management		
TEE_CheckMemoryAccessRights	Supported	
TEE_Free	Supported	
TEE_GetInstanceData	Supported	
TEE_Malloc	Supported	See section 6.2.2
TEE_MemCompare	Supported	See section 6.2.2
TEE_MemFill	Supported	See section 6.2.2
TEE_MemMove	Supported	See section 6.2.2
TEE_Realloc	Supported	
TEE_SetInstanceData	Supported	
Message Digest		
TEE_DigestDoFinal	Supported	
TEE_DigestUpdate	Supported	
Modular Arithmetic		
TEE_BigIntAddMod	Not Supported	
TEE_BigIntInvMod	Not Supported	
TEE_BigIntMod	Not Supported	
TEE_BigIntMulMod	Not Supported	
TEE_BigIntSquareMod	Not Supported	
TEE_BigIntSubMod	Not Supported	
Other Arithmetic		
TEE_BigIntComputeExtendedGcd	Not Supported	
TEE_BigIntIsProbablePrime	Not Supported	
TEE_BigIntRelativePrime	Not Supported	
Panic Function		
TEE_Panic	Supported	
Persistent Object		
TEE_CloseAndDeletePersistentObject	Supported	See section 6.2.4
TEE_CreatePersistentObject	Supported	See section 6.2.4
TEE_OpenPersistentObject	Supported	See section 6.2.4

TEE_RenamePersistentObject	Not Supported	
Persistent Object Enumeration		
TEE_AllocatePersistentObjectEnumerator	Not Supported	
TEE_FreePersistentObjectEnumerator	Not Supported	
TEE_GetNextPersistentObject	Not Supported	
TEE_ResetPersistentObjectEnumerator	Not Supported	
TEE_StartPersistentObjectEnumerator	Not Supported	
Property Access		
TEE_AllocatePropertyEnumerator	Not Supported	
TEE_FreePropertyEnumerator	Not Supported	
TEE_GetNextProperty	Not Supported	
TEE_GetPropertyAsBinaryBlock	Supported	See section 6.2.1
TEE_GetPropertyAsBool	Supported	See section 6.2.1
TEE_GetPropertyAsIdentity	Not Supported	
TEE_GetPropertyAsString	Supported	See section 6.2.1
TEE_GetPropertyAsU32	Supported	See section 6.2.1
TEE_GetPropertyAsUUID	Supported	See section 6.2.1
TEE_GetPropertyName	Not Supported	
TEE_ResetPropertyEnumerator	Not Supported	
TEE_StartPropertyEnumerator	Not Supported	
Random Data Generation		
TEE_GenerateRandom	Supported	
Symmetric Cipher		
TEE_CipherDoFinal	Supported	
TEE_CipherInit	Supported	
TEE_CipherUpdate	Supported	
TA Interface		
TA_CloseSessionEntryPoint	Supported	
TA_CreateEntryPoint	Supported	
TA_DestroyEntryPoint	Supported	

TA_InvokeCommandEntryPoint	Supported	
TA_OpenSessionEntryPoint	Supported	
Time		
TEE_GetREETime	Not Supported	
TEE_GetSystemTime	Not Supported	
TEE_GetTAPersistentTime	Not Supported	
TEE_SetTAPersistentTime	Not Supported	
TEE_Wait	Not Supported	
Transient Object		
TEE_AllocateTransientObject	Supported	See section 6.2.3
TEE_CopyObjectAttributes	Supported	
TEE_FreeTransientObject	Supported	
TEE_GenerateKey	Supported	
TEE_InitRefAttribute	Supported	
TEE_InitValueAttribute	Supported	
TEE_PopulateTransientObject	Supported	
TEE_ResetTransientObject	Supported	
Authenticated Encryption		
TEE_AEDecryptFinal	Supported	
TEE_AEEncryptFinal	Supported	
TEE_AEInit	Supported	

6.2.1 Properties

A property is an immutable value identified by a name, which is a Unicode string. The property value can be retrieved in a variety of formats: Unicode string, binary block, 32-bit integer, Boolean, and Identity.

At present <t-base supports the following subset of the Properties API:

- TEE and TA properties are supported, but no client properties.
- Properties must be accessed at their proper type; conversion to string is not supported.
- Properties cannot be enumerated.

The property set is passed to each function in a pseudo-handle parameter. The following table lists the defined and supported property sets:

Pseudo-Handle	Meaning

TEE_PROPSET_CURRENT_TA	The configuration properties for the current Trusted Application.
TEE_PROPSET_TEE_IMPLEMENTATION	The properties of the TEE Implementation itself

For `TEE_PROPSET_TEE_IMPLEMENTATION` property set the following properties are supported:

Property name	Property type	Property value
gpd.tee.apiversion	String	"1.0"
gpd.tee.description	String	Not defined yet
gpd.tee.deviceID	UUID	SUID obtained by tlApiGetSuid function

For `TEE_PROPSET_CURRENT_TA` property set the following properties are supported:

Property name	Property type	Property value
gpd.ta.appID	UUID	GP UUID of a given TA
gpd.ta.singleInstance	Boolean	False
gpd.ta.multiSession	Boolean	False
gpd.ta.instanceKeepAlive	Boolean	False
gpd.ta.dataSize	UINT32	Heap area size of a given TA
gpd.ta.stackSize	UINT32	Stack area size of a given TA

6.2.2 Memory Management

6.2.2.1 TEE_Malloc

```
DECLARE_TRUSTED_APPLICATION_MAIN_HEAP( uint32_t staticSize);

void* TEE_Malloc (
      size_t size,
      uint32_t hint );
```

To use TEE_Malloc, the developer must define the static size of the heap in the TA code through the DECLARE_TRUSTED_APPLICATION_MAIN_HEAP() macro. Otherwise the `TEE_Malloc` function returns NULL. The heap size is limited by the size of the TA address space. The `hint` parameter has to be 0, indicating a block of memory filled with zeros.

6.2.2.2 TEE_MemMove

The SDK uses the memmove() implementation of the compiler.

6.2.2.3 TEE_MemCompare

The SDK uses the memcmp() implementation of the compiler.

6.2.2.4 TEE_MemFill

The SDK uses the memset() implementation of the compiler.

6.2.3 Transient Objects

6.2.3.1 TEE_AllocateTransientObject

The following table lists the supported object types.**Error! Reference source not found.Error! ference source not found.Error! Reference source not found.Error! Reference source not found.Error! Reference source not found.**

Table 1: TEE_AllocateTransientObject: Supported Object Sizes

Object Type	Possible Object Sizes
TEE_TYPE_AES	128, 192, or 256 bits
TEE_TYPE_DES	Always 56 bits
TEE_TYPE_DES3	112 bits. No support for 168 bits
TEE_TYPE_HMAC_MD5	Not supported
TEE_TYPE_HMAC_SHA1	Between 80 and 512 bits, multiple of 8 bits
TEE_TYPE_HMAC_SHA224	Not supported
TEE_TYPE_HMAC_SHA256	Between 256 and 1024 bits, multiple of 8 bits
TEE_TYPE_HMAC_SHA384	Not supported
TEE_TYPE_HMAC_SHA512	Not supported
TEE_TYPE_RSA_PUBLIC_KEY	Key size up to 2048 bits
TEE_TYPE_RSA_KEYPAIR	Key size up to 2048 bits
TEE_TYPE_DSA_PUBLIC_KEY	Not supported
TEE_TYPE_DSA_KEYPAIR	Not supported
TEE_TYPE_DH_KEYPAIR	Not supported
TEE_TYPE_GENERIC_SECRET	Not supported

6.2.4 Persistent Objects

6.2.4.1 TEE_OpenPersistentObject

The completion of this function depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

Currently, the access permission setup for persistent storage object is not implemented. The storage file is accessible to all legal TA session regardless of its associated permissions.

6.2.4.2 TEE_CreatePersistentObject

The completion of this function depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

Currently, the access permission setup for persistent storage object is not implemented. The storage file is constructed with the `flags` passed as argument which will not constrain any file operations applied.

6.2.4.3 TEE_CloseAndDeletePersistentObject

This function does not destroy all opened session associated with the underlined storage file but the one passed as argument `object`. In case multiple copy of storage file are opened, the developer should consider the possibility of concurrency and keep their file object up-to-date.

6.2.5 Data Stream Access

6.2.5.1 TEE_ReadObjectData, TEE_WriteObjectData, TEE_TruncateObjectData

The completion of these functions depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

Currently, the access permission setup for persistent storage object is not implemented. The storage file is accessible to all legal TA session regardless of its associated permissions.

6.2.6 Generic Operation Functions

6.2.6.1 TEE_AllocateOperation

The following table lists the supported algorithms.

Table 2: TEE_AllocateOperation: Supported Modes

Algorithm	Possible Modes
TEE_ALG_AES_ECB_NOPAD TEE_ALG_AES_CBC_NOPAD TEE_ALG_AES_CTR	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT
TEE_ALG_AES_CTS TEE_ALG_AES_XTS TEE_ALG_AES_CCM TEE_ALG_AES_GCM	Not supported
TEE_ALG_DES_ECB_NOPAD	TEE_MODE_ENCRYPT

TEE_ALG_DES_CBC_NOPAD TEE_ALG_DES3_ECB_NOPAD TEE_ALG_DES3_CBC_NOPAD	TEE_MODE_DECRYPT
TEE_ALG_DES_CBC_MAC_NOPAD TEE_ALG_AES_CBC_MAC_NOPAD TEE_ALG_AES_CBC_MAC_PKCS5 TEE_ALG_AES_CMAC TEE_ALG_DES_CBC_MAC_PKCS5 TEE_ALG_DES3_CBC_MAC_NOPAD TEE_ALG_DES3_CBC_MAC_PKCS5	Not supported
TEE_ALG_RSASSA_PKCS1_V1_5_SHA1	TEE_MODE_SIGN TEE_MODE_VERIFY
TEE_ALG_RSASSA_PKCS1_V1_5_MD5 TEE_ALG_RSASSA_PKCS1_V1_5_SHA224 TEE_ALG_RSASSA_PKCS1_V1_5_SHA256 TEE_ALG_RSASSA_PKCS1_V1_5_SHA384 TEE_ALG_RSASSA_PKCS1_V1_5_SHA512 TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA224 TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA256 TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA384 TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA512 TEE_ALG_DSA_SHA1	Not supported
TEE_ALG_RSAES_PKCS1_V1_5 TEE_ALG_RSA_NOPAD	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT

Algorithm	**Possible Modes**
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1 TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224 TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256 TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384 TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512	Not supported
TEE_ALG_DH_DERIVE_SHARED_SECRET	Not supported
TEE_ALG_MD5	Not supported
TEE_ALG_SHA1 TEE_ALG_SHA256	TEE_MODE_DIGEST
TEE_ALG_SHA224 TEE_ALG_SHA384	Not supported

TEE_ALG_SHA512	
TEE_ALG_HMAC_SHA1 TEE_ALG_HMAC_SHA256	TEE_MODE_MAC
TEE_ALG_HMAC_MD5 TEE_ALG_HMAC_SHA224 TEE_ALG_HMAC_SHA384 TEE_ALG_HMAC_SHA512	Not supported

6.3 EXTENDED API

The following functions provide extended features and can be called from Trusted Applications developed with the TEE Internal API

6.3.1 Logging

6.3.1.1 TEE_LogPrintf

```
void TEE_LogPrintf (
      const char*      fmt,
      ...);
```

This proprietary function allows writing formatted traces for logging purposes.

6.3.1.2 TEE_LogvPrintf

```
void TEE_LogvPrintf (
      const char*      fmt,
      va_list args);
```

This proprietary function allows writing formatted traces for logging purposes.

6.3.1.3 TEE_DbgPrintf

```
void TEE_DbgPrintf (
      const char*      fmt,
      ...);
```

This proprietary function allows writing formatted traces for debugging purposes. It is only compiled in on debug builds of the TA.

6.3.1.4 TEE_DbgvPrintf

```
void TEE_DbgvPrintf (
      const char*      fmt,
      va_list args);
```

This proprietary function allows writing formatted traces for debugging purposes. It is only compiled in on debug builds of the TA.

6.3.2 TEE_TBase_UnwrapObject

```
TEE_Result TEE_TBase_UnwrapObject(
        void                *src,
        size_t              srcLen,
        void                *dest,
        size_t              destLen);
```

This function is equivalent to `tlApiUnwrapObject()` with flags set to `TLAPI_UNWRAP_PERMIT_DELEGATED`.

6.3.3 TEE_TBase_DeriveKey

```
TEE_Result TEE_TBase_DeriveKey(
        const void          *salt,
        size_t              saltLen,
        void                *dest,
        size_t             destLen);
```

This function is equivalent to `tlApiDeriveKey()` with context set to `MC_SO_CONTEXT_TLT` and lifetime set to `MC_SO_LIFETIME_PERMANENT`.

6.3.4 Trusted User Interface API

The Trusted User Interface API is also available for Trusted Applications developed with GlobalPlatform APIs.

The API is the same than the tlApi, only the names of the functions change:

```
TEE_Result TEE_TBase_TUI_GetScreenInfo(
     tlApiTuiScreenInfo_ptr screenInfo)

TEE_Result TEE_TBase_TUI_OpenSession(void)

TEE_Result TEE_TBase_TUI_CloseSession(void)

TEE_Result TEE_TBase_TUI_SetImage (
     tlApiTuiImage_ptr image,
     tlApiTuiCoordinates_t coordinates)

TEE_Result TEE_TBase_TUI_GetTouchEvent (
     tlApiTuiTouchEvent_ptr touchEvent)
```

Please refer to section 3.8 for details about the Trusted User Interface API.

6.3.5 DRM API

The Trusted User Interface API is also available for Trusted Applications developed with GlobalPlatform APIs.

The API is the same than the tlApi, only the names of the functions change:

```
TEE_Result TEE_TBase_DRM_OpenSession (
     int *sHandler);

TEE_Result TEE_TBase_DRM_ProcessContent (
    uint8_t                         sHandle,
    tlApiDrmDecryptContext          decryptCtx,
```

```
    uint8_t                             *input,
    tlApiDrmInputSegmentDescriptor       inputDesc,
    uint16_t                            processMode,
    uint8_t                             *rfu);

TEE_Result TEE_TBase_DRM_CloseSession (
      int sHandler);

TEE_Result TEE_TBase_DRM_CheckLink (
      uint8_t sHandler,
      tlApiDrmLink_t link)
```

Please refer to section 3.9 for details about the Trusted User Interface API.